(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: 0 363 173 A2

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 89310147.7

(51) Int. Cl.5: G06F 13/12 , H04L 29/08

(22) Date of filing: 04.10.89

(30) Priority: 07.10.88 US 254986

(43) Date of publication of application:
11.04.90 Bulletin 90/15

(84) Designated Contracting States:
DE FR GB IT

(71) Applicant: International Business Machines Corporation
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: Davis, Gordon Taylor
1285 West Royal Palm Road
Boca Raton Florida 33486(US)
Inventor: Mandalia, Baiju Dhirajlal
9900 Baywater Drive
Boca Raton Florida 33496(US)
Inventor: Landa, Robert Egugene
7159 Northwest 3rd Avenue
Boca Raton Florida 33487(US)
Inventor: Van den Berg, Jan Wouter
541 Phillips Drive
Boca Raton Florida 33432(US)
Inventor: Van Voorhis, David Curtis
1225 Southwest 21st Street
Boca Raton Florida 33486(US)

(74) Representative: Grant, Iain Murray
IBM United Kingdom Limited Intellectual
Property Department Hursley Park
Winchester Hampshire SO21 2JN(GB)

(54) **Communication processors for layered protocols.**

(57) A special purpose communications protocol processor (CPP) provides more efficient processing of layered communication protocols -- e.g. SNA (Systems Network Architecture) and OSI (Open Systems Interconnection) -- than contemporary general purpose processors, permitting hitherto unavailable operations relative to high speed communication links. The CPP contains special-purpose circuits dedicated to quick performance (e.g. single machine cycle execution) of functions needed to process header and frame information, such functions and information being of the sort repeatedly encountered in all protocol layers, and uses instructions architected to operate these circuits. The header processing functions given special treatment in this manner include priority branch determination functions, register bit reshaping (rearranging) functions, and instruction address processing functions. Frame processing functions so handled include CRC (cyclic redundancy check) computations, bit insertion/deletion operations and special character detection operations.

EP 0 363 173 A2

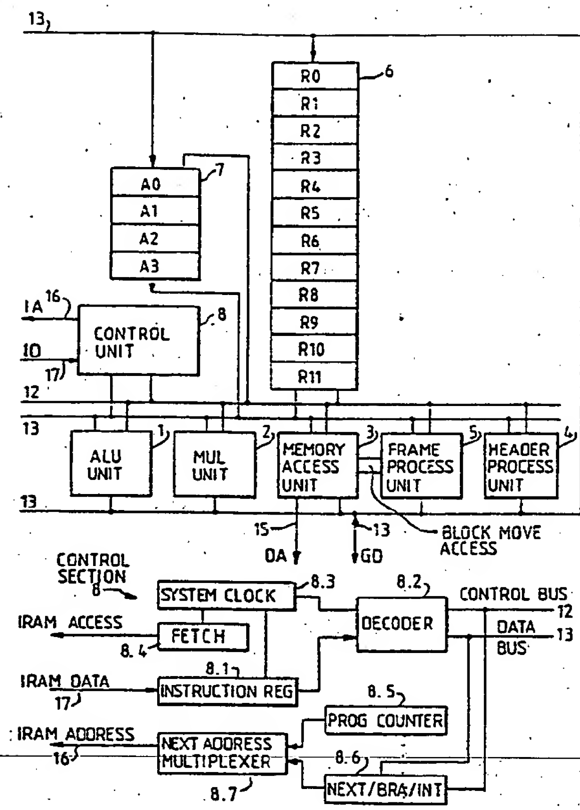

13
R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
6
A0
A1
A2
A3
7
IA
16
CONTROL UNIT
8
IO
17
12
13
ALU UNIT
1
MUL UNIT
2
MEMORY ACCESS UNIT
3
FRAME PROCESS UNIT
5
HEADER PROCESS UNIT
4
15
13
DA
GD
BLOCK MOVE ACCESS
CONTROL SECTION
8
SYSTEM CLOCK
8.3
8.2
DECODER
CONTROL BUS
12
DATA BUS
13
IRAM ACCESS
FETCH
8.4
8.1
IRAM DATA
17
INSTRUCTION REG
8.5
PROG COUNTER
IRAM ADDRESS
16
NEXT ADDRESS MULTIPLEXER
8.6
NEXT/BRA/INT
8.7

FIG. 1

COMMUNICATION PROCESSORS FOR LAYERED PROTOCOLS

The present invention relates to communication processor for layered protocols or, in other words, to computer architecture particularly applicable to communications controllers. CCITT protocols supported by Open Systems Interconnection (OSI) standards are considered a basis for the architecture underlying the present invention, although other protocols such as SNA could be equally supported. Specific device areas to which this invention may be beneficially applied include network interfaces, front end communication controllers for large systems and telecommunications switching nodes. This architecture is also applicable to network gateway implementations and protocol conversion.

The use of programmable communications controllers to concentrate signal traffic between multiple communications lines and data processing equipment is not per se new. U.S. Patents 3,863,226; 4,156,796; 4,156,932; 4,188,665; 4,328,543; and 4,484,263 disclose various methods of implementing such controllers, using commercially available general purpose microprocessors as processing engines, and "off-the-shelf" asynchronous receiver and transmitter (USART) devices.

Related co-pending patent application 89307144.9 discloses a hardware assist mechanism to handle the media access (MAC) layer for protocol flexibility. It recognises a need for special purpose terminal equipment to support various types of information transmission needs of the future.

However, we are unaware of any recognition in the art of the particular problem addressed here; i.e. the need for dealing specifically with header and frame processing functions repeatedly encountered in layered communication protocols, at speeds and protocol levels adequate for present and future needs in high throughput real time communication. This problem is solved presently by providing special purpose circuits and associated instructions dedicated to speeding up handling of such repeatedly encountered header and frame processing functions at all levels of todays layered protocols (e.g. OSI standards). These circuits and instructions include implementational features permitting execution of associated header and frame processing operations in single machine cycles, with flexibility suited to accommodating similar operations at the physical, data link, network, transport, session, presentation and application layers of OSI and other layered communication protocols. Those skilled in the art will recognise that the prior art above uses communication control processors which are directed to the handling of link access functions and generally rely upon more powerful processors in intermediate or host systems for processing communication protocol layers above the link access level.

More specifically, our present understanding is that contemporary general purpose microprocessors and digital signal processors, as used to implement processing engines or CPU (Central Processing Unit) elements in prior art communication controllers, were not considered suitable for handling support functions associated with processing of header and frame information in layers of today's layered communications protocols above the link control or media access layers; nor has the art apparently recognised that a need for such support exists at the link access level. We have found however that such support is needed for efficient use of high speed media (T-1, T-2, fiber optics, etc.) Stated otherwise, we have recognised that processing loads imposed by present and future high speed links are such that, without offloading of header and frame process functions above the link control layer, I/O and processing bottlenecks would inevitably occur, with consequent blockage or reduction of throughput speeds limiting effective media usage.

Of course, there have been major advances in areas related to this work; e.g. improvements in microprocessor and VLSI technologies applicable to communication controls, advances in protocols per se and improvements in parallel processing. Although these might tend to alleviate the presently recognised problem, they do not provide the more effective and complete solution to which the present invention is directed.

The present invention provides a communication processor adapted to execute layered communication protocols, wherein instructions of first and second types are stored in memory; the instructions of the first type designating predetermined operations associated with functions particular to processing header and frame information elements in each of a plurality of layers in a layered communication protocol and which would, if decoded and executed via general purpose logic circuitry, take multiple machine cycles to complete, instructions of the second type designating operations for performing functions other than the predetermined functions; the processor including

special purpose circuit means responsive to instructions of the first type for performing the entire operation designated by each respective instruction in a single the machine cycle;

general purpose arithmetic logic circuit means responsive to instructions of the second type for performing functions designated by respective instructions; and

means responsive to a common partial decode of each invoked instruction to route appropriate control

signals to either the appropriate special purpose circuit means or to the general purpose circuit means.

In other words, as embodied in an example thereof disclosed hereinafter, it can produce a special purpose Communications Protocol Processor (CPP), and a newly defined instruction architecture therefore, having dedicated units or circuit elements for implementing key header and frame processing functions frequently encountered in all layers of today's layered communication protocols. Such dedicated units perform their respective operations faster than contemporary microprocessors at modest cost compared to benefits. In addition to these special units, the CPP contains a general purpose arithmetic logic unit (ALU). The special units and associated registers provide single machine cycle execution of header and frame information processing operations which otherwise would take multiple machine cycles for execution in the ALU and which we recognise presently as major processing loads needing special handling.

Header functions receiving special attention in this manner include:

- Programmable Priority branch on bit operations:

Branch on bit of immediate data is known, but does not provide either selection of the order of priority given to the bits which represent branching conditions or parallel evaluation of bits. US patent 4,573,118 discloses a branch on bit operation with parallel evaluation of condition bit priority, but without presently featured capability for varying the priority ordering under user program control.

- Register reshape (Shift, mask, and swap) operations:

Shifting, masking, and swapping are also known per se, but the present invention improves by performing all three functions with a single instruction, and in a single machine cycle of the system.

- Address processing operations:

Present circuit uses instructions with associatively matched addressing for translating routing information in headers, thereby speeding translation operations and reducing storage requirements for target address information.

- Frame functions receiving similar treatment are:

-- CRC (Cyclic Redundancy Check) computations:

Specialised Cyclic Redundancy Check (CRC) circuits to support quick execution of error checking for various protocol. This would enable support for standard 16 and 32 bit CRC polynomials. Most standard protocols, including HDLC, DDCMP and BISYNC, use CRC checks. These circuits are programmable to set up any polynomial.

-- Bit insertion/deletion operations:

Specialised insertion/deletion circuits provide transparency for data relative to special control characters in each protocol layer. Such transparency is needed in protocols like HDLC to isolate special control characters, denoting start and end of frame, from actual data.

-- Special character detection operations:

Dedicated circuits, used with a special instruction, provide character oriented protocol functions, as needed for most character oriented protocols including BISYNC and DDCMP.

By relegating execution of the above functions to a programmable set of newly architectured instructions acting through special units, as we have done, each unit can be used to provide the function

capabilities of many universal asynchronous communications controllers (UACCs), while maintaining flexibility to handle diverse communication lines and protocols. The CPP also includes a special set of registers dedicated for data link control operations. The functions supported enhance processing of data in standard link level protocols; e.g. HDLC, X.25, SNA, TCP, etc.. State of the art pipelining and parallelism, block move, and cycle steal memory access techniques have been adopted to further enhance performance.

Above-mentioned block move techniques allow for concurrent transfer of blocks of information and processing of portions of the same information, as well as concurrent execution of other operations such as CRC (Cyclic Redundancy Check) computations, and bit insertion/deletion.

The CPP has the ability to perform up to three operations in parallel executing with a three phase pipeline. Separate instruction and data busses permit pipelining of data references with instruction fetches. The operations may be any from the following categories.

1. Arithmetic and register transfer operations
2. Multiplier operations
3. Memory Access operations
4. Header processing for data protocols
5. Frame processing for data protocols

The CPP contains four indexing address registers, twelve general purpose data registers and other special purpose protocol support registers. All registers are 32 bits wide. The processor can support 64K Words of instruction memory and 16MBytes of data memory. Figure 3 shows a typical environment for using the advantages of subject processor. Summarising features and capabilities of this architecture: * In addition to parallelism and pipelining for speeding up retrieval of instructions, throughput is further enhanced by providing for execution of the newly defined instructions in single system machine level cycles. Pipelining and parallelism are illustrated in Figure 2. Care has been taken in the design of the present special purpose units to consider data and branch dependencies and provide programming capabilities to enhance performance.

* Memory load/store, data shifting, multiplication and other ALU operations are done in parallel and in single processor cycles.

The present invention also provides the counterpart methodology.

ADDITIONAL REFERENCES

(1) Hill, F., Peterson, "Digital Systems Hardware Organization and Design", Prentice Hall. 1987.

(2) Beaven, P. A., "Parallel Programmable Array Structured CRC generator", IBM Technical Disclosure Bulletin, Oct 1978, pp2058-2062.

(3) Hwang, Kai and Briggs, F. A. " Computer architecture and parallel processing", McGraw Hill, 1984. (4) Tanenbaum, A. "Structured Computer Organization", Prentice Hall, 1984

The present invention will be described further by way of example with reference to embodiments thereof and contrasted examples of the prior art, as illustrated in the accompanying drawings, in which:-

Figure 1 is a block diagram of the CPP system illustrating associated registers and special purpose processing units;

Figure 2 illustrates prior art techniques for parallelism and pipelining to speed up processing;

Figure 3 illustrates a typical environment in which the CPP is expected to operate;

Figure 4 illustrates CPP instruction formats;

Figure 5 illustrates the CPP header processing unit;

Figure 6 illustrates a CPP unit for executing branch on priority bit detect (BDD) instructions as presently defined;

Figure 7 illustrates a priority address mechanism in the unit of Figure 6;

Figure 8 illustrates an address stack array addressed by the priority encoder of Figure 7;

Figure 9 shows details of the priority encoder of Figure 7;

Figure 10 illustrates a register reshape unit for executing register reshape (RSH) instruction operations;

Figure 11 illustrates a swap and rotate multiplexer in the reshape unit;

Figure 12 shows a swap unit element of the reshape unit;

Figure 13 illustrates another embodiment of the reshape unit allowing for multiple parameter extraction;

Figure 14 illustrates an address routing unit for executing operations called for by Address Routing (ARS) instructions;

Figure 15 illustrates elements of the address route unit;

Figure 16 illustrates the memory access unit of the CPP;

Figure 17 illustrates a block move unit for the CPP;

Figure 18 illustrates a controller for block move operation;

Figure 19 illustrates a frame processing unit for the CPP;

Figure 20 illustrates a unit for single cycle execution of cyclic redundancy check (CRC) computations;

Figure 21 illustrates a unit for bit insertion/deletion operations;

Figure 22 shows details of the bit processing unit in Fig. 21;

Figure 23 illustrates a unit for branch on control character detect operations; and

Figure 24 shows details of the unit shown in Figure 23.

CPP ARCHITECTURE

Figure 1 shows the overall architecture of subject communications protocol processor (CPP). Arithmetic logic Unit (ALU) 1 provides basic arithmetic operations of ADD, SUBTRACT and COMPARE, logical operations of OR, AND, and XOR (Exclusive-Or), and various register transfer operations. Multiplier unit (MUL) 2 supports 16 by 16 multiplication with scaling and rounding functions in conjunction with ALU outputs. Memory access unit 3, header processing unit 4, and frame processing unit 5 -- constituting special elements of the CPP for performance of key functions in accordance with the present invention -- are described in detail later. General purpose data register stack 6 and address index register stack 7 comprise portions of internal variably allocatable storage that can be used as input and output for processing units 1 - 5 via control bus 12 and data bus 13.

Control Unit 8, containing elements 8.1 - 8.7 shown at the lower portion of Figure 1, controls and monitors operation execution. Instruction register 8.1 receives instructions from instruction memory (IRAM) shown elsewhere. Instruction operation codes (opcodes) in this register are decoded by decoder 8.2 which generates control signals to process units 1 - 5 through control bus 12 and causes immediate data or address information to be transferred via data bus 13. System clock 8.3 provides timed control signals to the other elements. Fetch control 8.4 at clocked intervals directs instruction fetching action relative to IRAM.

Data bus 13 contains multiple bi-directional conduction paths allowing for parallel transfer of data between register stacks 6 and 7 and process units 1 - 5. Program counter 8.5 and next branch (or interrupt) control logic 8.6 generate next addresses one of which is selected by multiplexer 8.7 for application to IRAM via IRAM address bus 16. Data address (DA) 15 is used to connect memory access unit 3 to Data Ram (DRAM, also shown elsewhere), for causing transfer of data via data bus (GD) 13.

Parallel execution of operations is possible using pipelining techniques suggested at 20 and 21 in Figure 2. While instruction 'n' is being executed, a next instruction 'n+1' is decoded and the instruction following it, instruction 'n+2', is fetched, as shown at 20. In addition, if three operations are called for by one instruction, their decoding and execution may be performed in parallel as suggested at 21. Bus transfer mechanisms and pipelining techniques for such operations are well known and described in "additional references" (1), (3), and (4) above.

Figure 3 illustrates a typical environment in which this CPP, shown at 31, would function advantageously. DRAM 32 interfaces with multiplexor and input/output interface 33 to exchange data with physical communication interface 34. IRAM 35 stores instructions of application programs for directing the CPP to perform functions dynamically required at interface 34. Host interface 36 provides for exchange of systematic handshake signals needed to sustain bi-directional transfer of data and commands between the CPP and a host processing system. Block 37 represents layered protocol applications and signal processing functions which can be integrally accommodated in this environment.

Instruction Format Definition

The instruction set was designed with a fixed format and instruction length of 32 bits. Figure 4 shows the detailed definition of bit assignments in the instruction words. A typical 32 bit instruction 40 consists of four parts or fields; a format definition field and three opcode fields, the latter calling for up to three discrete operations. Format definition field 41 consist of four parts or subfields. A 2-bit subfield 42 specifies one of three types of instruction formats as follows:

- 00 - 3 parallel operations

* 01 - 2 parallel operations with optional register and address operands
* 10 - 1 branch operation with extended addressing

Subfields 43-45, 2-bits in each, specify operations in associated categories for corresponding opcode fields (1-3). Categories within each field are arithmetic computation, memory access, frame processing and header processing.

Typical opcode field 46 designates an operation via 5-bit operation subfield 47 and 3 bit register field 48.

In addition to the special purpose CPP instructions, the instruction set comprises general purpose instructions for designating ALU, MULTIPLY, and MEMORY LOAD/STORE operations, using techniques described in "additional reference" (4) by Tanenbaum.

Figures 6 to 14 discussed later illustrate special purpose units for performing the operations designated by subject CPP instructions. The instruction set comprises instructions with associated operations as follows.

Instruction Set

1) ALU and Memory Load store operations.

FMn RR - load register RR From Memory using index An
TMn RR - store register RR To Memory using index An
RTn RR - transfer register n to RR
ARn RR - add register n to RR
SRn RR - subtract register n from RR

2) Frame Processing operations.

* CRC RR - Compute CRC on register RR
* BSW RR - perform bit insertion/deletion using RR
* BCD RR - branch if RR has a special character

3) Header Processing operations.

* BBD RR,X - branch on priority bit detect in RR, X is the stack selection parameter.
* RSH RR - reshape register RR
* ARS RR - compute new address using RR

4) Block Move operation

* BMO RR - perform block move with queue command in RR

5) Parallel operations

** EP3 OP1<RR{,OP2<RR{,OP3<RR{ - three parallel operations
** EP2 OP1<RR{,OP2<RR{ - two parallel operations
** EP1 OP1<RR{ - one operation

Register Structure

This table shows register assignments and functions.

REGISTER	FUNCTION
_A0 to A3	_ MEMORY ACCESS AND INDEXING REGS. _
_R0 to R11	_ GENERAL PURPOSE DATA REGISTERS _
_ CRCP	_ CRC POLYNOMIAL REGISTER _
_ CRCA	_ CRC ACCUMULATOR REGISTER _
_ BSHP	_ BIT TRANSPARENCY PATTERN REGISTER _
_ BSHS	_ BIT TRANSPARENCY STATUS REGISTER _
_ RSHC	_ RESHAPE CODE REGISTER _
_ SRC	_ SOURCE ADDRESS FOR BLOCK MOVE _
_ DST	_ DESTINATION ADDRESS FOR BLK. MOVE _
_ CNT	_ BUFFER LENGTH FOR BLOCK MOVE _
_ LCIR	_ LOGICAL CHANNEL ID FOR ADDRESS _

The following sections describe circuits for executing Instructions marked "*".

Header Processing Unit

Figure 5 shows components of a header processing unit 4 (see previous discussion of Figure 1). Branch On Priority Bit Detect (BBD) unit 60 provides a mechanism responsive to associated BBD instructions (see list above) to implement the operations for branching program execution as a function of condition bits designating branch conditions, and featuring priority selection of the bits representing active conditions based on priority selection data which can be programmably preloaded into registers in this unit. ~~This unit is described and separately claimed in co-pending application (BC9-88-009 cross-referenced).~~

Register reshape unit 61 rearranges registered data to extract required parameters in a format suited for evaluation. Address routing unit 62 translates header address information as needed for establishment of connections and network routes through the various protocol layers.

BBD Unit

As shown in Figure 6, information in input latch 80, representing various system conditions and/or interruption request events, is processed in accordance with a predetermined priority ordering of respective latched bits, for branching the program execution of the system to a next instruction address representing the beginning of a program routine for dealing with the selected condition. Priority select mechanism 81 determines which of the bits in latches 80 representing active conditions (requiring branching action) has highest priority and operates to select an instruction address from associated address register stack shown elsewhere for transfer to output register 82. Stack initialise register 83 is used to initialise address stack elements (shown elsewhere) in the priority address mechanism 81. Decoders 84 and 85, responsive to the BBD instruction opcode on control bus 12, generate control signals for gating input latch 80 to the priority address mechanism and for gating outputs of that mechanism to branch address register 82. Decoder 84 responsive to instructions designating register transfer operation generates control signals to load stack initialise register 83 via data bus 13.

Figure 7 describes the priority mechanism 81 of Figure 6. A feature of this mechanism is the programmability of its priority ordering function as determined by priority select stack 102. Values programmably loaded into stack 102 determine the relative priorities of active condition bits in latches 80, and actuate the priority encode unit 103 to select an address from branch address stack 104 in access decode mechanism 105 in response to the BBD instruction. The selected address, associated with the active bit in latches 80 having highest priority value set in its associated register within stack 102, is transferred to output register 82 as the address of the next instruction to be executed; thereby initiating a branch routine suited to the selected branch condition.

This priority mechanism is further described and separately claimed in the cross-referenced application for "Programmable Priority Branch Circuits". Another feature of this mechanism is that it is arranged to perform its foregoing address extraction operation (to locate the selected branch target instruction), in a single machine cycle of the CPP system. Stacks 102 and 104 can be initialised via data path 107, in response to standard block move instructions. As another feature, these stacks can be provided with

multiple segments relative to latches 80, and thereby consolidate loading operations for establishing multiple sets of priority and address associations relative to bit positions in latches 80. Such consolidation provides the additional benefit of permitting selection of stack segments by BBD instructions, so that priority ordering and/or branch address associations may be dynamically varied without repeated loading of either stack (e.g. to support multiple protocols); thereby further improving system adaptability and throughput.

Figure 8 shows the mechanism to access branch address stack 104; the latter shown in this figure at 116 and its input path 107 shown here at 115. To initialise stack 116, decoder 118 (corresponding to decoder 84 in Figure 6) processes information from register loading instructions to gate data from bus 115 to a selected register in the stack. To read an address from the stack, another decoder (not shown, but corresponding to decoder 85 in Figure 6) reacts to the highest priority select value in stack 102 (Figure 7), in response to a BBD instruction, to select output from an associated register in stack 116 for transfer to output register 82 via output data bus 119. As noted earlier, in order to support plural protocols, stack 116 may contain plural segments, each loadable with a different set of branch addresses, for relating different sets of branching functions to sets of conditions in input latch 80. This requires providing information in BBD instructions for designating selection of different segments.

Details of a programmable priority encoder unit 103 are shown in Figure 9. Bits from input latch 80 (Figure 7), labled bit 1, bit 2, ... , bit 16, serve as selection control inputs relative to associated priority select registers 120a through 120d (bit 1 activating 120a, bit 2 activating 120b, ... , and bit 16 activating 120d). Priority select code values in registers 120a through 120d (each 4-bits) may be initialised in response to register loading instructions of the processor, such as Data Move instructions (Register to Register or Memory to Register) or Load Immediate instructions. Multiple registers 120x may be initialised in parallel since the processor data bus will typically carry many more bits than can be used in a single priority select register.

As noted previously, the purpose of initialising the priority registers 120x is to allow for programmably varying priorities associated with the condition bits in input latch 80. Since there are 16 condition bits in the illustrated embodiment, 4 bits are needed per priority select register to be able to assign unique priority select rankings to all condition bits. In general, N priority select bits would be needed relative to a set of condition bits having a length of 2 to the Nth power.

In response to BBD instructions, values loaded into registers 120x associated with input bits in active state, are gated out to respective logic circuits 122x through 127x. The latter operate in effect to compare received magnitudes and gate as final output the largest magnitude; i.e. the highest priority value. Thus, for instance, if the highest priority select value is held in register 120b, and bit 2 is on or active, the value in 120b will be gated through the array 122x -127x, regardless of the other values held in registers 120.

With input latches 80, registers 120x and address stack 116 (Figure 8) pre-loaded, the circuit is prepared for execution of the BBD instruction operation. In response to such instruction, array 122x -127x in Figure 9 effectively selects the highest priority select value in registers 120x associated with the respective input latch bit in active condition representing the highest priority branch condition or event. Bits of the selected value are gated to outputs P0, P1, P2, and P3 (P3 being the most significant bit) which act through a not-shown decoder (corresponding to decoder 85 in Figure 6) to address output of an associated register in stack 116 (Figure 8) and thereby transfer the respective address to output register 82.

More specifically, the operation of the array of gates proceeds as follows: active or "ON" input bits -- Bit 1, Bit 2, ..., Bit 16 -- from input latch 80 enable outputs of associated priority select registers 120x to be transferred to associated rows of gates 122x - 127x. Input latch bits which are inactive or "OFF" will force outputs of the associated priority register to be "OFF". Outputs from priority registers 120x are compared by respective rows of gates, starting with the most significant bit. OR gate 129a examines the most significant bits of the active priority registers 120x, and sets output P3 "ON" if at least one of its inputs is "ON". Output P3 feeds back to Compare gates 122a, 122b, 122c.... Output of each Compare gate 122x will be turned "ON" only if the most significant bit from the corresponding priority register 120x is at the same logic level ("ON" or "OFF") as Encoder output P3. All input bits for which the corresponding Compare gate outputs are "OFF" are eliminated from further consideration in determining the other encoder outputs P2, P1, and P0 since an "OFF" state at the output of Compare gate 122a, 122b ... causes AND gates 123a, 125a, 127a, 123b, 125b, 127b, ... to block further propagation of priority register 120a, 120 b ... outputs. In like manner, each lower order bit from each of the priority registers 120x is processed, but only after degating those register outputs eliminated in processing the previous priority bit. Thus outputs of AND gates 123a, 123b ... correspond to the second to the most significant bit for those Inputs Bit 1, Bit 2 ... which are active and which have a priority number with the most significant bit set to the same state as Output P3. OR gate 129b generates Encoder Output P2 based on inputs from AND gates 123a, 123b .... Compare gates 124a, 124b ... eliminate additional priority numbers for which the second bit does not match,

and AND gates 125a, 125b ... for remaining priority numbers drive OR gate 129c to generate Encoder output P1. In a like manner, Compare gates 126a, 126b ... eliminate additional priority numbers for which the third bit does not match, and AND gates 127a, 127b ... for remaining priority numbers drive OR gate 129d to generate Encoder output P0.

As mentioned previously, outputs P0 - P3 are used to address the address register stack with an encoded 4-bit value which represents a highest active branch condition. Alternately, outputs of AND gates 127a, 127b, ..., could each be associated directly to registers in the register stack and could, when activated, gate the output of the corresponding address register to the address bus. This is possible because only one of the AND gate outputs 127x may be activated at any one time, unless two priority registers are loaded with the same number (something which should not be done). This alternative would prioritise the address selection in accordance with positions of the active input latch bits per se, rather than numbers programmably associated with those bits. Note that in the alternative configuration the lowest priority input latch bit, that in latch position 1 for instance, would be given priority only if no other bit is active. Since only 15 non-zero selection functions can be formed from 16 latch bit positions, simple logic can be added to check for a condition of all latch bits equal 0 and in that circumstance gate an output for selecting a branch target address associated with absence of active conditions, if a unique branch in that circumstance is required.

The Register Reshape Instruction Circuit

Circuits for execution of the Register Reshape (RSH) instruction are shown in Figure 10. Information in input latch 174 is reshaped as per the code In Reshape (RS) code register 175 by operation of Register reshape unit 176. Output is sent to output register 177. The RS code register is initialised via data bus 13 with a suitable shift, mask and swap operation before execution of associated RSH operations. Decoders 178 and 179, responsive to the RSH instruction opcode on control bus 12, generate control signals for gating input latch 174 to register reshape unit 176 and also for gating the output from the reshape unit register 177. Decoder 178 responsive to a register transfer operation via data bus 13 generates the control signal to load register 175 with data designating the reshape options.

Reshape options that can be coded into register 175 (assuming a 32 bit input register) include:

1) Swap 8 least significant bits
2) Swap 16 least significant bits
3) Swap 32 least significant bits
4) Rotate N bits
5) Mask M most significant bits

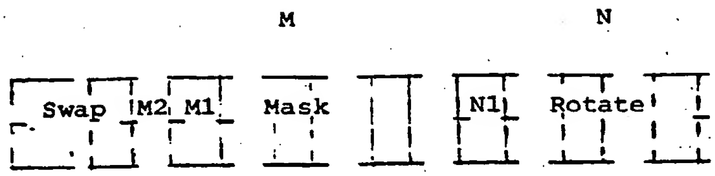
SWAP L bits here means bits D(O) to D(L), in the input register, will be interchanged according to the relationships:

$$
\begin{aligned}
D(L) &= D(0) \\
D(L-1) &= D(1) \\
\cdot \quad & \quad \cdot \\
\cdot \quad & \quad \cdot \\
D(1) &= D(L-1) \\
D(0) &= D(L)
\end{aligned}
$$

ROTATE N Bits above means bits in the input register are shifted right or left N bit places per:
Shift right N means $D(i) = D(i+N)$; i is the ith bit
Shift left N means $D(i) = D(i-N)$; i is the ith bit

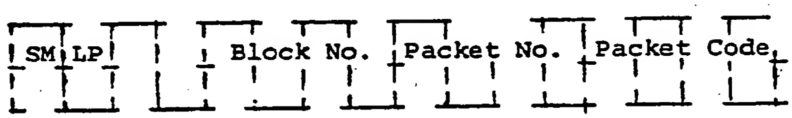
MASK M bits above means set a specified number of bits to 0 or 1, beginning with the most significant or least significant bit.

The reshape code register to satisfy the above options would appear as below.

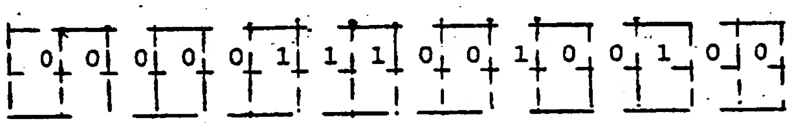


N1 indicates direction of shift
M1 indicates set or reset for Mask
M2 indicates from MSB or LSB for Mask M and N represent number of bits, maximum 32
Swap Code
001 - Least 8 bits
010 - Least 16 bits
011 - All 32 bits
100 - Most Significant 8 bits
101 - Middle 16 bits
110 - All 32 bits

Example: In the register shown above, assume we need to extract the packet number



The reshape code would be,



N = 4 shift four bits
N1 = 1 shift direction right
M = 28 Mask 28 bits
M1 = 0 Reset Mask for 0
M2 = 0 Start Mask from MSB
Swap Code = 000 No Swap

AHPL (A hardware programming language) has been used here to define the logic of the reshape unit. Described below is AHPL data for the reshape register construct. Chapter 5 of "Additional Reference" (1) discusses details of this language.

```
MODULE:  RESHAPE REGISTER
 INPUTS:  resdecode    */ res instruction decode line
    Rn          */ any register n
    RSHC     */ reshape code register -175
 OUTPUTS:Rm[16]
         */ any register m
```

```
BUSSES:SWAPOUT[32];
1.  Rm < - RSH[Rn].
```

The RSH computation logic contains two stages as shown in Figure 11. The first stage consists of a set of 32 multiplexers 209, each with inputs from latch 174 relatively shifted in position by one bit. Swap units 210 (32 of them) use the shift code and the swap code from the reshape code register (Block 175 in Figure 10) to provide different selection control inputs to the multiplexers 209. The second stage consists of the mask logic 212 that is controlled by the mask code from reshape code register 175 (Figure 10) to provide the proper set/reset to the sequence of bits desired. Mask logic 212 can be described by the following relations.

Let output word vector D(L) be of length L bits, let M specify mask code, M1 specify set/reset, and M2 specify right/left. Then D(i) = 1 if M1 = 1 for i { M if M2 = 0
or i < M if M2 = 1
or
D(i) = 0 if M1 = 1 for i { M if M2 = 0
or i < M if M2 = 1

For all other values D(i) will be the same as the ith multiplexer 209 in Figure 11.

Figure 12 shows details of the swap unit. Decoder 219 uses the swap code to select one of the AND gates 217 to provide the desired range of bit swap; 8, 16 or 32 bits. Compare logic 216 uses decoder 219 output in conjunction with register 218 and output of the associated multiplexer number 214 to determine whether the swap unit is to perform a swap function relative to that multiplexer. The shift code is exclusive-ORed with outputs of AND gates 217 at 215 to provide the multiplexer output address which selects 1 of 32 of the multiplexer inputs to be the output. XOR gates 215 complement the least significant 3, 4 or 5, bits of the shift code depending on the swap code requested.

Figure 13 is an extension of the reshape mechanism for allowing extraction of multiple parameters at a time. This variation uses a reshape code register with the following functions to provide a speedier operation. Input register 201 is common for all but there are three output registers 204 processed by three reshape units 203. The reshape code register 202 would then contain three separate codes to extract the three parameters (one for each reshape unit).

The ARS Instruction Circuit

This circuit is shown in Figure 14. Information in input latch 252 is processed for address translation. The logical channel ID 253 is used by address generator 254 to generate the actual routing address 255. Decoders 256 and 257, responsive to the ARS instruction opcode on control bus 12, generate control signals for gating input latch 252 to the address route generator 254 and also to latch the output to the route address 255. Decoder 256, responsive to a register transfer operation via data bus 13, generates the control signal to load the logical channel ID 253.

Described below is an AHPL data notation description for the logic to control address routing operation. Figure 15 describes the hardware used for the operation. The address stack 261 and the route stack 262 are initialised with address conversion parameters with one to one correspondence. This allows compare circuit 263 in conjunction with priority encoder circuitry 264 to use the input register Rn 260 value to select a corresponding routing address in the route stack 262 which can be sent to the output register (Block 255 in Figure 14). A stack access decode mechanism similar to the one in Figure 8 is used to initialise the address and route address stacks.

```
MODULE: ADDRESS ROUTE INSTRUCTION
  MEMORY: ADDRSTK<64[[16] */ address stack 64 words
    ROUTSTK<64[[16]       */ route stack 64 words
  INPUTS: arsdecode       */ ARS instruction decode line
           Rn                */ any register
  OUTPUTS:Rm             */ any register
  BUSSES: CMPBUS[8]      */ compare bus
  1. Rm <- ARS[Rn].
   ARS(Rn) = DCD(PRIORITY ENCODE (CMPBUS)) * ROUTSTK
     */ PRIORITY ENCODE is the standard priority encode
   CMPBUS(64) = (V(Rn.XOR.ADDRSTK(1)))__ */ compare
           (V(Rn.XOR.ADDRSTK(2)))__      */ characters
             (V(Rn.XOR.ADDRSTK(3)))__
               (V(Rn.XOR.ADDRSTK(4)))__
                .......................
                 (V(Rn.XOR.ADDRSTK(61)))__
                (V(Rn.XOR.ADDRSTK(62)))__
                  (V(Rn.XOR.ADDRSTK(63)))__
                    (V(Rn.XOR.ADDRSTK(64)))
```

Memory Access Unit

Figure 16 provides a block schematic of the memory access unit which consists of the block move unit 295 and memory load store unit 292. Memory load store unit 292 provides standard data memory access as in commonly implemented microprocessors within prior art as in reference (4).

The BMO (Block Move) Instruction Circuit

This circuit is shown in Figure 17. Source address register 346 can be loaded by a register transfer instruction with the start address location of a data block. Block move status register 351 is updated by monitor unit 350 as the move proceeds. The block move circuit includes a mechanism to transfer data between source and destination addresses with data translation. Source address 346 is the first memory location where source data resides. Destination address 347 is the first memory location to which data is sent. The buffer length, which is the number of data bytes to be transferred from memory beginning at the source address, can be stored in the count register 348.

The circuit permits block move requests to be queued in a stack. The queue command register 349 inserts the contents of 346, 347 and 348 into the queue. It also permits the user to specify if data translation is desired (e.g. for cyclic redundancy check or bit insertion/deletion) in conjunction with the block move so that the paths 354 to the frame process unit (Block 5 in Figure 1) are enabled. The registers 346, 347, 348 and 349 can be loaded with a register transfer operation using data bus 13. Decoders 352 and 353, responsive to block move instruction opcode on control bus 12, generate the control signals for gating registers 346, 347 and 348 for loading into the block move circuit 350 using the queue command register. Decoder 352 responsive to a register transfer operation. via data bus 13 generates the control signal to load from the block move circuit 350 the block move status register 351 which indicates the status of the block

move function. The data address path 15 is used for addressing the external DRAM (Block 32 Figure 3) and internal stacks (Figures 7, 15 and 23).

Described below is an AHPL sequence for constructing the controls of the block move logic 361. Figure 18 describes details of circuits used in the AHPL sequence. Different sets of block move requests are stored in first come first served order using queue 360. Queue pointer 363 points to location of request currently processed by the block move circuit. Software queue pointer 367 points to the last request submitted to the queue for block move action. Block move status 351 will indicate when the queue 360 becomes empty or full, depending on positions of the hardware and software queue pointers 363 and 367. The data translation registers for bit insertion/ deletion 369 and cyclic redundancy check 368 are used when frame processing is enabled relative to path 354.

Multiplexers 435 in Figure 20 and 459 in Figure 21 are selected by decoders 432 and 456 respectively to enable path 354 to use processing units 430 and 454 respectively to compute the new Cyclic redundancy code (CRC) 368 and bit insertion/deletion word (BSW) 369. It should be noted that block move actions occur only when DRAM is accessible on a cycle steal basis (other units not requesting access), and when data translation is requested with the block move then the transfer waits for a machine cycle when the frame process unit is not being used by other instructions.

```
MODULE: BLOCK MOVE
  MEMORY: SRCADDR[16];  */ register for source address
          DESTADDR[16]; */ register for destination
                           address
          COUNT[16];    */ register for buffer length
                           for block move
          QPTR[16];     */ block move queue pointer
          SQPTR[16];    */ software queue pointer
          BMOSTAT[16];  */ block move status
          DATA[32];     */ data register
          QUEUE<64[[16];*/ queue stack
          ACT[1]        */ activity status bit
          CRCA[32]      */ CRC accumulator
          BSWA[32] */ Bit insertion/deletion status/data
  INPUTS: cyst          */ cycle steal control line
          crc           */ frame process for CRC enable
          bsw           */ frame process for BSW enable
  OUTPUTS: qdone        */ qdone signal
  BUSSES: DATADDR[16];CDATA[16]  */ address and data
                                   busses for data ram
```

```
 1. -{_ACT[1]/1      */ test activity - queue contents
 2. SRCADDR[16]  <-  DCD(QPTR)*QUEUE   */ load block move
                                          information
    QPTR<-INC(QPTR)
 3. DESTADDR[16]  <-  DCD(QPTR)*QUEUE
    QPTR<-INC(QPTR)
 4. COUNT[16]  <-  DCD(QPTR)*QUEUE
    QPTR<-INC(QPTR)
 5. BMOSTAT[16]  <-  DCD(QPTR)*QUEUE
    QPTR<-INC(QPTR)
 6. -{cyst/6                   * scan for cycle steal signal
 7. CDATA <- DCD(SRCADDR)*MEMORY  * perform data move
    DCD(DESTADDR)*MEMORY <- CDATA
    SRCADDR <- INC(SRCADDR)
    DESTADDR <- INC(DESTADDR)
    COUNT <- DEC(COUNT)
 7.1 -{_crc/7.3               * test if CRC in parallel
 7.2 CRCA <- CRC(CDATA)       * perform CRC operation
 7.3 -{_bsw/8                 * test if BSW in parallel
 7.4 BSWA <- BSW(CDATA)       * bit insert/delete
 8. -{(V/COUNT)/6             * is count zero
 9. -{(V/(QPTR.XOR.SQPTR))/2  * is queue empty
10. qdone=1 -{ 1
```

Frame Processing Unit

Figure 19 shows components of the frame process unit dedicated to performing operations on frame information and packets for overall error checking and data transparency. The cyclic redundancy code unit 389 performs error checking via cyclic redundancy code (CRC) computations. The bit insertion/deletion hardware unit 390 provides insertion and deletion of bits on invocation of desired data patterns for separating frame delimiters and control words from actual data. The branch on control character detect unit 391 provides an automatic branching mechanism on special control characters for speeding up decision functions.

The CRC (Cyclic Redundancy Check) Instruction Circuit

This circuit is described by Figure 20. Information in register 428 is processed for CRC computation 430 using the polynomial register RR 429. The output is available in CRC accumulator 431. Register 429 is a 32 bit register programmable for any standard CRC-CCITT or CRC-16. It is important that the polynomial register and CRC accumulator be initialised before each new computation. The control bus provides proper flow of data. Registers 429 and 431 can be loaded by a register transfer operation using data bus 13. Decoders 432 and 433, responsive to the CRC instruction opcode on control bus 12 generate the control signals for gating the input latch 428 to CRC computation unit 430 and also to latch the new accumulated

value to CRC accumulator 431. Decoder 432, responsive to a register transfer operation via data bus 13, generates the control signal to load the CRC polynomial register 429. Multiplexer 435 and demultiplexers 434, 436, serve to select the path of usage of the CRC computation unit 430, either for a regular instruction or as part of a data translation accompanying a block move operation. The computation logic 430 is non clocked and has been mentioned in prior art reference Beaven 78, in the IBM Technical Disclosure bulletin Vol. 21, No. 5, October 1978, pages 2058-2061, Reference (2).

The BSW Instruction Circuit

This circuit is described by Figure 21. Information in register 452 is processed for bit insertion/deletion 454 using pattern register RR 453. The output is available in the status - output byte 455. The pattern register is an eight bit register containing a sequence of bits on the occurrence of which the output may insert or delete a bit for transparency. It is important that the pattern register and the output status byte be initialised before any new computation. The control bus provides proper flow of data. The registers 453 and 455 can be loaded by a register transfer operation via data bus 13. Decoders 456 and 457, responsive to the BSW instruction opcode on control bus 12, generate control signals for gating input latch 452 to bit processing unit 454 and also to latch the new byte and status to status-output 455. Decoder 456, responsive to a register transfer operation via data bus 13, generates the control signal to load pattern register 453. Multiplexer 459 and demultiplexers 458, 460 are serve to select the path of usage of bit processing unit 454 either for a regular instruction or as part of a data translation for block move operation.

Figure 22 shows details of bit processing unit 454 of Figure 21. Data from input latch 452 and status word 455, also in Figure 21, is selected by select units 480 (S) so that result data from each unit consists of six bits relatively shifted by one bit for all input lines. Compare units 481 (C) compare each six bit value from units 480 with input from pattern register 453 (Figure 21), and outputs are fed to bit insert/delete logic 482. The table at the lower part of Figure 22 shows the function of logic 482 whose output includes byte OB and status for register 455 (Figure 21); the status consisting of the previous byte PB, the residue bits after insertion or deletion, the count of bits, and a flag E denoting existence or non-existence of residue value of 8.

The BCD Instruction Circuit

This circuit is described by Figure 23. Information in register RR 556 is processed for character detection. An automatic branch is taken to the new branch address output 559. Decode load register 557 is used to initialise branch on character unit 558 for the various branch addresses. Decoder initialisation is dependent on the protocol being used and the branch addresses needed for decision branching routines. The control bus provides proper flow of data. Decoders 560 and 561, responsive to the BCD instruction opcode on control bus 12, gate input latch 556 to branch on character unit 558 and latch the new branch address 559. Decoder 560 responsive to a register transfer operation via data bus 13 generates control signals to initialise the elements of 558.

Described below is an AHPL data notation description for logic of the control character detect operation. Figure 24 describes hardware used for the operation. The character stack 566 and the branch address stack 567 are initialised with addresses for conversions with one to one correspondence. This allows compare circuit 568 in conjunction with priority encoder circuitry 569 to use the value in input register Rn 565 to select a corresponding branch address in branch address stack 262 which can be sent to the output register (Block 559 in Figure 23). The stack access decode mechanism is as shown in Figure 8 and as described in detail relative thereto in terms of the priority branch on bit detect function.

```
MODULE: BRANCH ON CHARACTER DETECT
  MEMORY: CHARSTK<8[(8)   */ character stack 8 bytes -566
          ADDRSTK<8[(16)  */ branch address stack 8 words
                             (16b) -567
          BR(16)          */ branch register -559
  INPUTS: bcddecode       */ BCD instruction decode line
          Rn              */ any register
  OUTPUTS:BR(16)           */ branch register - 559
  BUSSES: CMPBUS(8)        */ compare bus
  1. BR <- BCD(Rn).

   BCD(Rn) = DCD(PRIORITY ENCODE (CMPBUS)) * ADDRSTK
   */ PRIORITY ENCODE is standard priority encode
   CMPBUS(8) = (V(Rn.XOR.CHARSTK(1)))__  */ compare
              (V(Rn.XOR.CHARSTK(2)))__     */characters
                (V(Rn.XOR.CHARSTK(3)))__


                (V(Rn.XOR.CHARSTK(4)))__
                  (V(Rn.XOR.CHARSTK(5)))__
                 (V(Rn.XOR.CHARSTK(6)))__
                   (V(Rn.XOR.CHARSTK(7)))__
                     (V(Rn.XOR.CHARSTK(8)))__
```

Claims

1. A communication processor adapted to execute layered communication protocols, wherein instructions of first and second types are stored in memory; the instructions of the first type designating predetermined operations associated with functions particular to processing header and frame information elements in each of a plurality of layers in a layered communication protocol and which would, if decoded and executed via general purpose logic circuitry, take multiple machine cycles to complete, instructions of the second type designating operations for performing functions other than the predetermined functions; the processor including

special purpose circuit means responsive to instructions of the first type for performing the entire operation designated by each respective instruction in a single the machine cycle;

general purpose arithmetic logic circuit means responsive to instructions of the second type for performing functions designated by respective instructions; and

means responsive to a common partial decode of each invoked instruction to route appropriate control signals to either the appropriate special purpose circuit means or to the general purpose circuit means.

2. A processor as claimed in claim 1, wherein the special purpose circuit means comprises separate circuit units for processing header information and frame information.

3. A processor as claimed in claim 2, wherein the header processing circuit unit comprises separate units for performing in single the machine cycles: priority branching operations, register reshape operations and address processing operations; the priority branching operations serving to test header bits denoting

branch conditions in a predetermined order of priority ranking and to produce as end result an address of an instruction suited to the conditions found in the test; the register reshape operations being useful for manipulating header data for reordering or masking bit sequences; and the address processing operations serving to translate addresses in information frames to corresponding route and/or memory addresses for further transfer of information.

4. A processor as claimed in claim 3, wherein the priority branching unit comprises:
an input register for storing a predetermined number n of data bits carried in the header segments; the bits individually when active denoting conditions requiring special branching action;
a stack of n address registers for storing addresses associated with individual bit positions in the input register;
priority encode circuitry coupled to the register and stack for selecting an address from the stack associated with a highest priority ranked bit in the input register having a value denoting an active condition requiring branching action; and
means for presenting the selected address to the memory means for designating the next instruction to be fetched therefrom.

5. A processor as claimed in claim 3 or claim 4, wherein the register reshape circuit unit comprises:
an input register for storing a predetermined number n of data bits carried in the header segments; the bits having predetermined positional ordering and carrying variable header information;
a reshape code register for defining options for swapping, masking, and shifting relative to data in the input register;
means for multiplexing and decoding the input register and the reshape code register to extract information logically related to the contents of the input register in accordance with functions defined in the reshape code register;
an output register; and
means for transferring the information extracted by the multiplexing and decoding means into the output register.

6. A processor as claimed in claim 3, claim 4 or claim 5, wherein the unit for performing address processing operations comprises:
an input register for storing a predetermined number n of data bits carried in the header segments; the bits carrying address information for information data frames;
a channel identification register for defining logical channels to which values in the input register belong;
a stack of n addresses for associative lookup of data relating to the data in the input register to generate route addresses
a comparator and priority encoder to select the appropriate route address and store it in the destination address register.

7. A processor as claimed in any of claims 2 to 6, wherein the frame processing circuit unit comprises separate circuit units for performing cyclic redundancy check computations, bit insertion/deletion operations and character detection operations.

8. A processor as claimed in claim 7, wherein the unit for performing cyclic redundancy check (CRC) operations comprises:
an input register containing data on which CRC computation is to be performed;
a polynomial code register which contains a polynomial used for the CRC computation;
a CRC computation unit for performing a CRC computation on data in the input register in a single the machine cycle, using the polynomial stored in the polynomial register; and
a CRC accumulator register for receiving results of the CRC computation.

9. A processor as claimed in claim 7 or claim 8, wherein the unit for performing bit insertion/deletion operations comprises:
an input register for data on which bit insertion or deletion operations are to be performed;
a pattern register for storing a pattern used for determining where to insert or delete data;
a bit insertion/deletion unit for operating in a single the machine cycle to insert bits into or delete bits from data received from the input register, at positions determined by the pattern register, the unit performing its operation in a single the machine cycle; and
an output register for storing result data produced by the bit insertion/deletion unit.

10. A processor as claimed in any of claims 7 to 9, wherein the unit for performing character detection operations comprises:
an input register for storing a byte of data carried in the header segments;
a stack of n branch addresses; and
a comparator/priority encoder to select one of the branch addresses from the stack in association with the

data in the input register.

11. A processor as claimed in any preceding claim, wherein the memory access unit means comprises: block move circuit means including means to transfer data between source and destination addresses with data translation; the source address designating a first memory location where source data resides; the destination address designating a first memory location for storing destination data;
means to store buffer length in a count register; the buffer length denoting a number of data bytes to be transferred from memory beginning at the source address;
means to queue block move requests in a stack; the queue having capacity to store a plurality of block move requests on a first come first served basis using queue pointers; the queue pointers comprising both hardware based and software based pointer elements; the hardware based elements denoting location of a request currently being processed by the block move circuit; the software based elements denoting location of a last request submitted to the queue for block move;
the data translation means designating bit insertion/deletion or cyclic redundancy check in conjunction with block move operation;
the means to transfer data comprising a block move control unit for executing the queue requests in predetermined order.

12. A processor as claimed in any of claims 3 to 11, wherein the header processing unit includes a priority branch unit for performing the priority branch operations, the priority branch unit comprising:
input latches for storing n condition bits defining active and inactive branching conditions;
means responsive to a BBD (Branch on Bit Detect) instruction for loading the latches with one of k sets of n condition defining bits;
an address stack comprising k sets of n addresses representing destinations determinable by bits in respective the sets of condition bits; and
means responsive to the BBD instruction for selecting a set of addresses from the stack associated with the set of condition bits currently in the latches and for selecting from the selected set of addresses an address associated with a bit in the latches; the associated address designating location of the next instruction to be fetched for execution in the system and the associated bit denoting an active condition having highest priority for attention.

13. A processor as claimed in claim 12, wherein the means for selecting the address sets and address comprises:
a stack of k sets of n selection control registers, each set associated with a respective one of k sets of condition bits loadable into the input latches; and
a priority encoder circuit coupled to the control register stack and the input latches, the encoder circuit responding to a the BBD instruction to select one of the k sets of selection control registers associated with bits currently latched in the input latches and to further select one control register in the selected set in association with a highest priority active bit in the latches; and
means responsive to contents of the selected selection control register to select an associated one of the addresses in an associated one of the k sets of addresses in the address stack.

14. In a communication network, a method of processing header and frame information accompanying message data units in each layer of a layered communication protocol, the header information comprising first, second and third information units, the first unit comprising a set of bits representing status of a respective set of communication conditions which when active require attention in a predetermined order of priority, the second unit comprising address information for determining routing of the accompanying message unit, and the third unit comprising a set of bits representing message variable parameters associated with respective message units, the third set requiring rearrangement in form for accurate extraction of the associated information, the frame information in each layer comprising check characters requiring cyclic redundancy check (CRC) computation processing and control characters uniquely demarking frame positions and requiring that other information in the respective units be subjected to bit insertion/deletion and processes in order to ensure unique recognisability of the control characters, the method comprising:
processing information in multiple the layers at a single processing centre at each node of the network; each centre having a predetermined machine cycle period of operation; each centre including a general purpose processing section and a plurality of special purpose processing sections; each special purpose section designed for performing a complex operation in a single the machine cycle; and
concentrating complex processing operations relative to each of the header units as well as the CRC computation and the insertion/deletion and functions into the special purpose sections, whereby respective header unit operations and CRC computations and bit insertion and deletion operations are each performed within single the machine cycle periods.

15. The method of claim 14 wherein the concentrated handling of the first set of header unit information includes:
within one of the machine cycle periods, performing a priority encoding operation on the bits of the first set to produce a priority code indication associated with a highest priority one of the bits representing an active condition requiring special action; and
within the same cycle period and as a function of the associated priority code indication locating an associated first instruction of a program routine for effecting the special action.

16. The method of claim 14 wherein the concentrated handling of the second set of header unit information includes:
within a single the machine cycle period, extracting an address associated with the second header unit by performing an associative table lookup translation.

17. The method of claim 14 wherein the concentrated handling of the third set of header unit information includes:
preloading the information of the third set into a register; and
within a single the machine cycle period performing a selected one of a plurality of register reshaping functions on the information to present the information in a required rearranged format.

18. The method of claim 14 wherein the concentrated handling of the frame unit information includes:
performing each CRC operation in a single the machine cycle period.

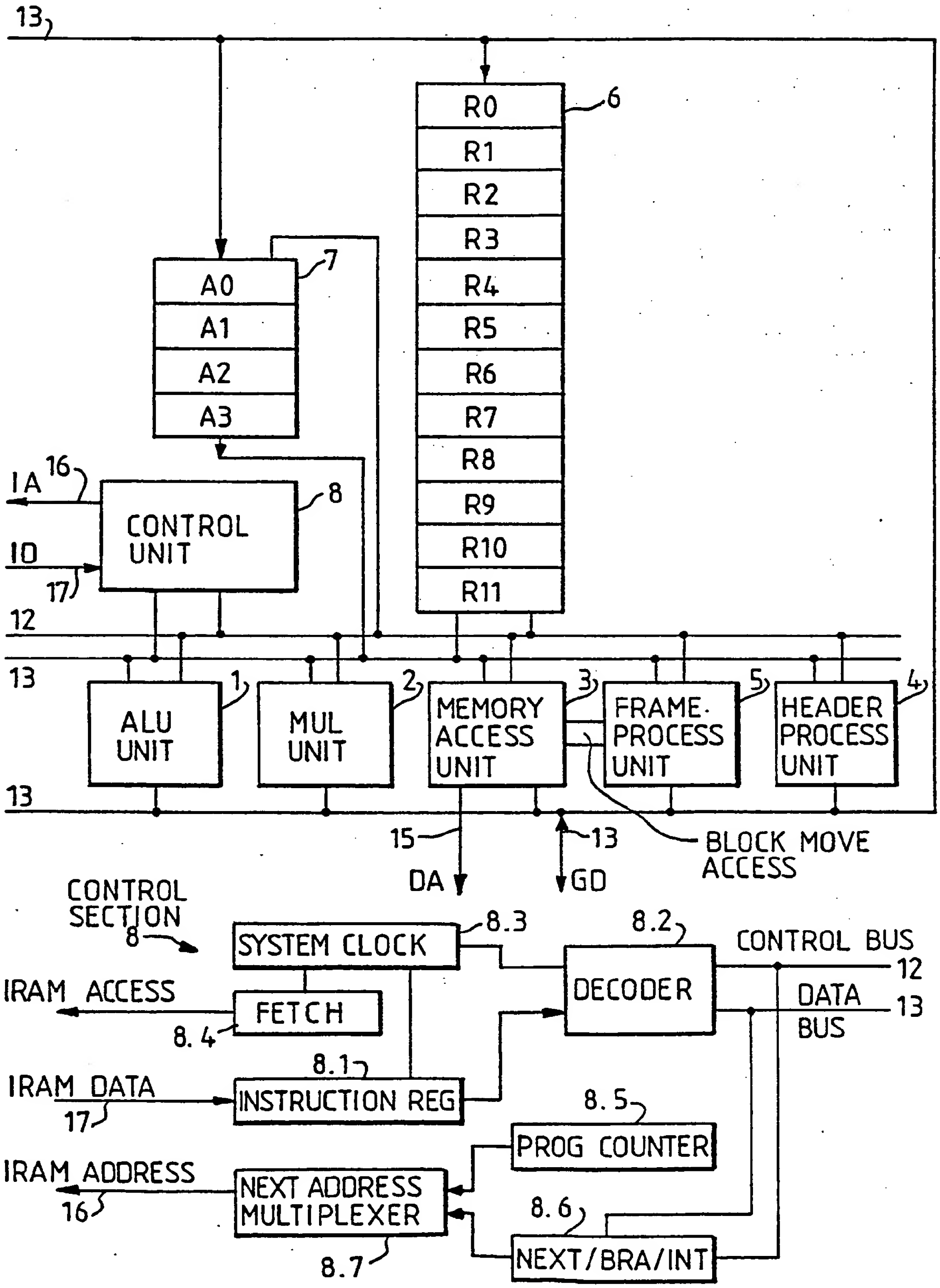

13
R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
6
A0
A1
A2
A3
7
IA
16
CONTROL UNIT
8
ID
17
12
13
ALU UNIT
1
MUL UNIT
2
MEMORY ACCESS UNIT
3
FRAME PROCESS UNIT
5
HEADER PROCESS UNIT
4
13
15
13
BLOCK MOVE ACCESS
DA
GD
CONTROL SECTION
8
SYSTEM CLOCK
8.3
8.2
DECODER
CONTROL BUS
12
DATA BUS
13
IRAM ACCESS
FETCH
8.4
8.1
IRAM DATA
INSTRUCTION REG
17
8.5
PROG COUNTER
IRAM ADDRESS
NEXT ADDRESS MULTIPLEXER
16
8.6
NEXT/BRA/INT
8.7

FIG. 1

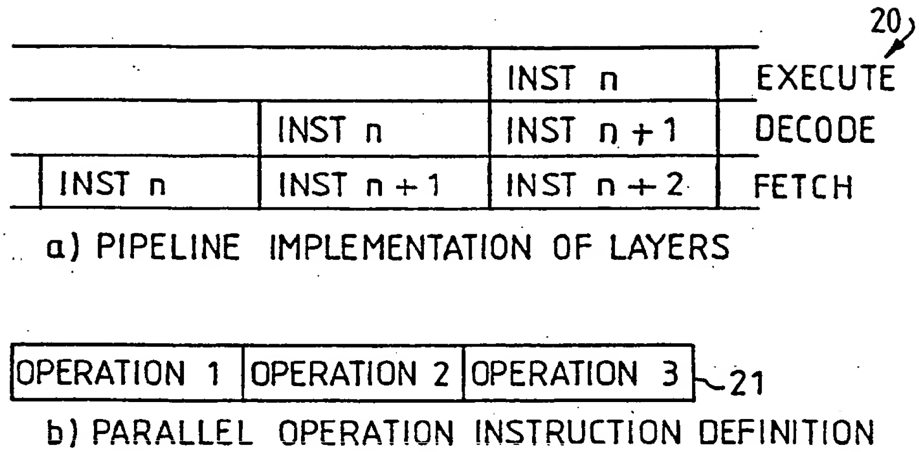

20
INST n
EXECUTE
INST n
INST n + 1
DECODE
INST n
INST n + 1
INST n + 2
FETCH
a) PIPELINE IMPLEMENTATION OF LAYERS
OPERATION 1
OPERATION 2
OPERATION 3
21
b) PARALLEL OPERATION INSTRUCTION DEFINITION

FIG. 2

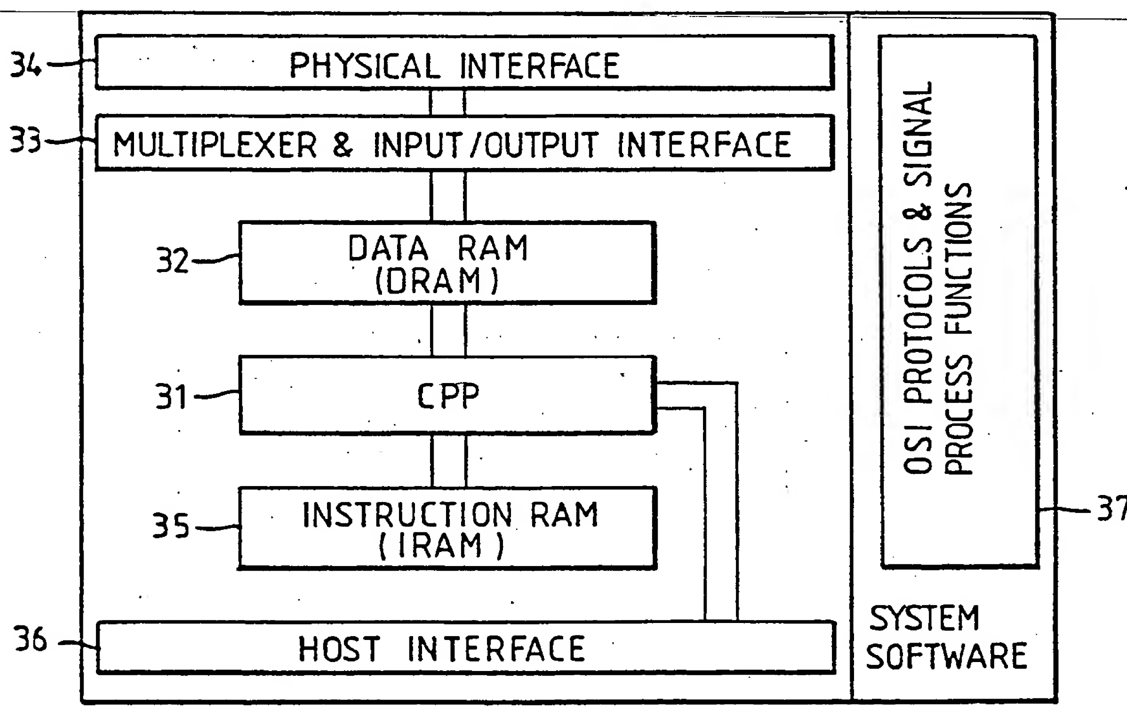

34
PHYSICAL INTERFACE
33
MULTIPLEXER & INPUT/OUTPUT INTERFACE
32
DATA RAM
(DRAM)
31
CPP
35
INSTRUCTION RAM
(IRAM)
36
HOST INTERFACE
OSI PROTOCOLS & SIGNAL
PROCESS FUNCTIONS
37
SYSTEM
SOFTWARE

FIG.3

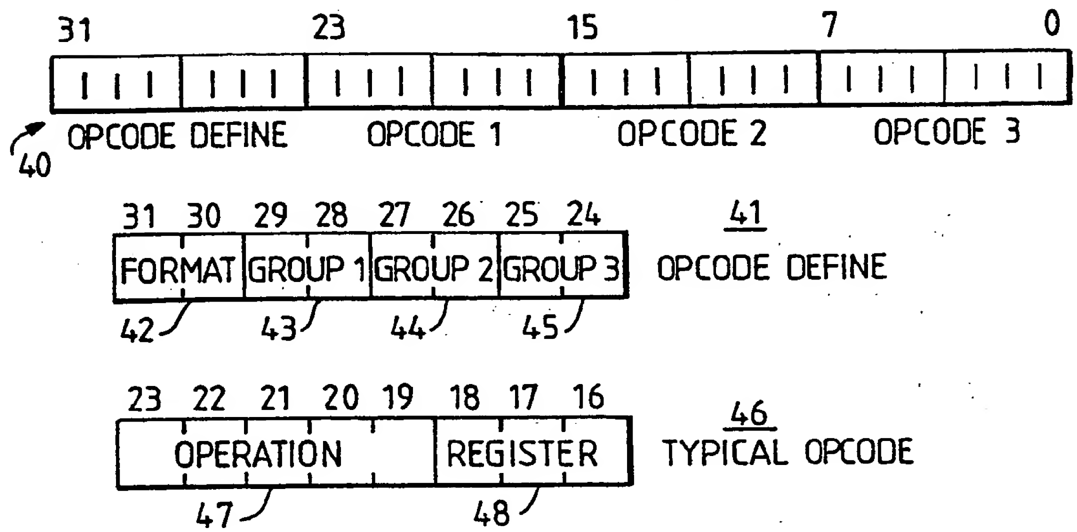

31
23
15
7
0
OPCODE DEFINE
OPCODE 1
OPCODE 2
OPCODE 3
40
31 30 29 28 27 26 25 24
41
FORMAT
GROUP 1
GROUP 2
GROUP 3
OPCODE DEFINE
42
43
44
45
23 22 21 20 19 18 17 16
46
OPERATION
REGISTER
TYPICAL OPCODE
47
48

FIG. 4

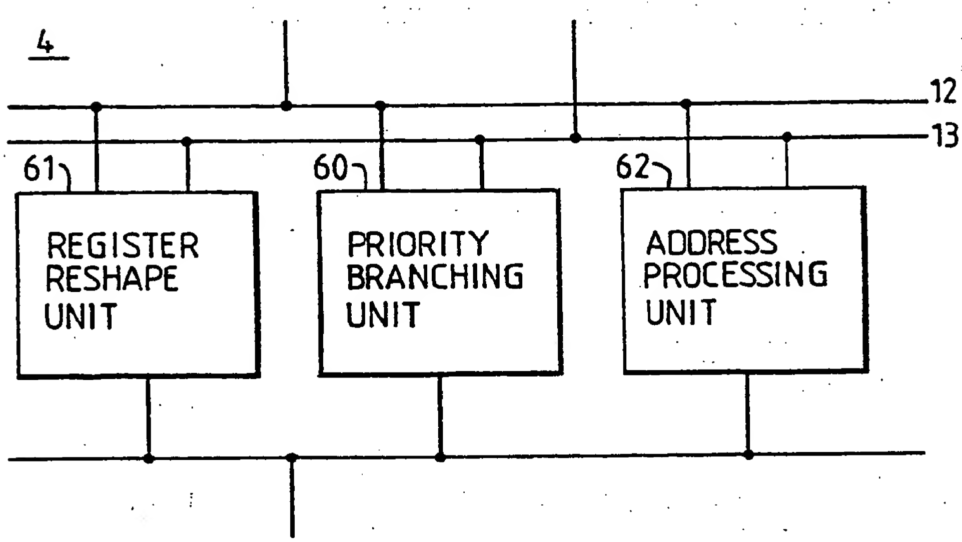

4
12
13
61
60
62
REGISTER RESHAPE UNIT
PRIORITY BRANCHING UNIT
ADDRESS PROCESSING UNIT

FIG. 5

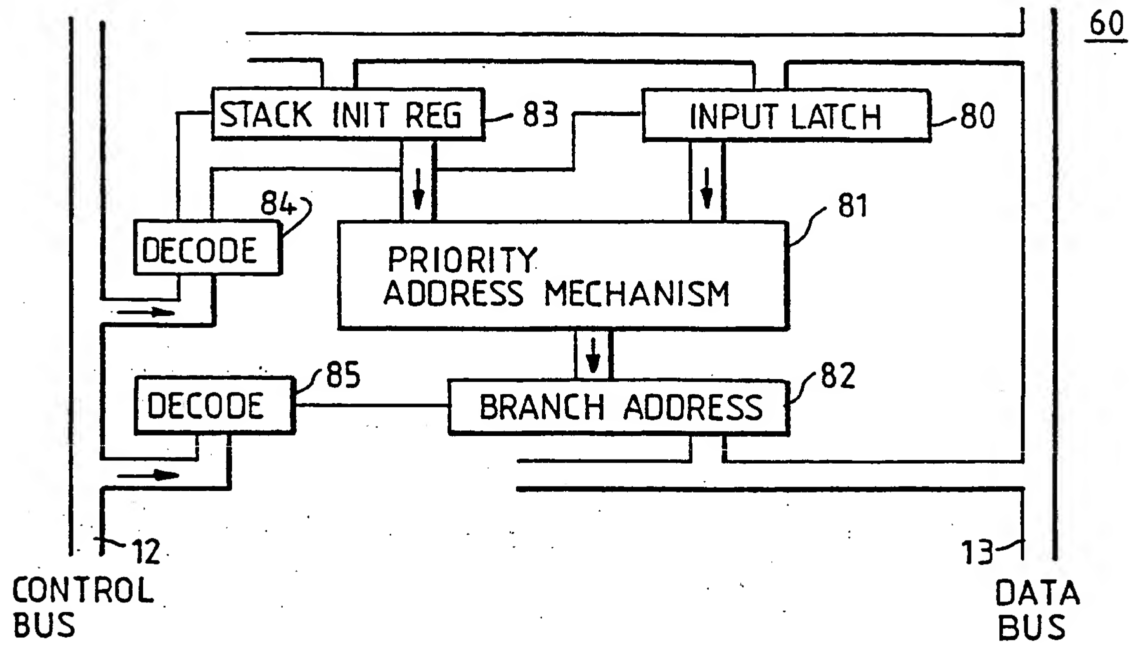

60
STACK INIT REG
83
INPUT LATCH
80
84
DECODE
81
PRIORITY
ADDRESS MECHANISM
85
DECODE
82
BRANCH ADDRESS
12
CONTROL
BUS
13
DATA
BUS

FIG. 6

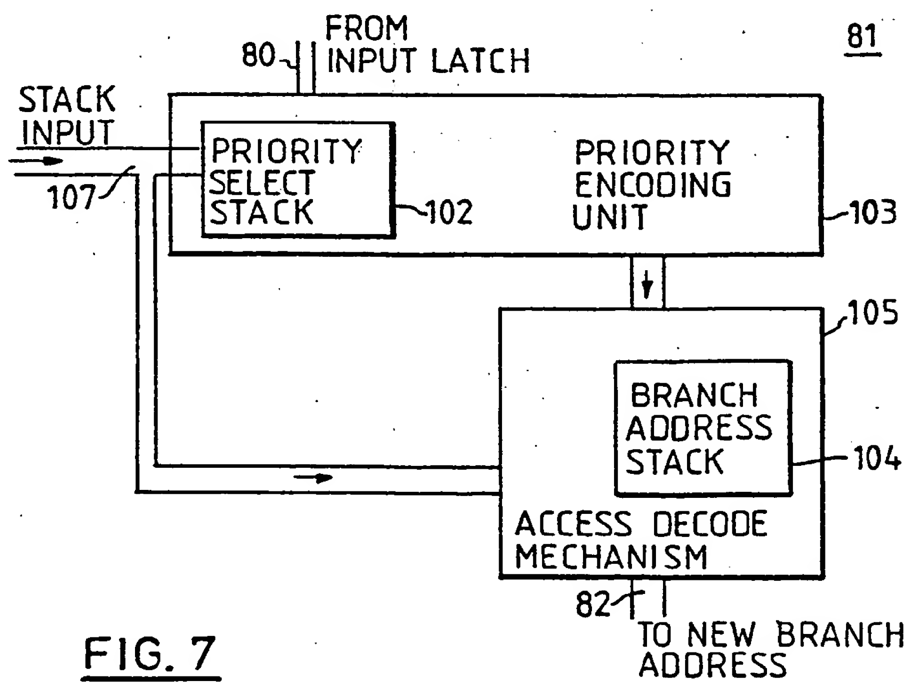
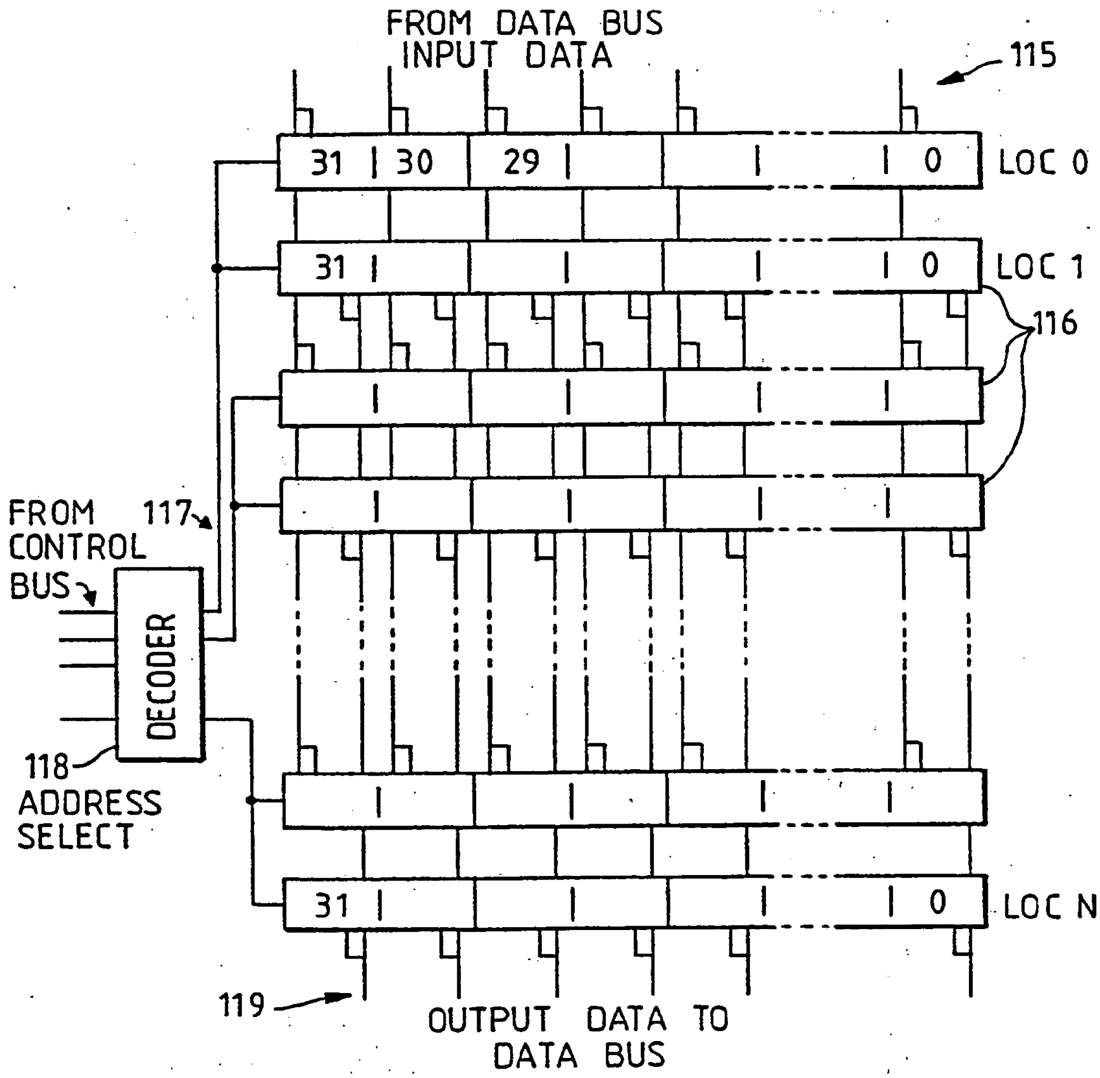

FROM
INPUT LATCH
80
81
STACK
INPUT
PRIORITY
SELECT
STACK
PRIORITY
ENCODING
UNIT
107
102
103
105
BRANCH
ADDRESS
STACK
104
ACCESS DECODE
MECHANISM
82
TO NEW BRANCH
ADDRESS

FIG. 7

FIG. 8


FROM DATA BUS
INPUT DATA
115
31
30
29
0
LOC 0
31
0
LOC 1
116
FROM CONTROL BUS
117
DECODER
118
ADDRESS SELECT
31
0
LOC N
119
OUTPUT DATA TO
DATA BUS

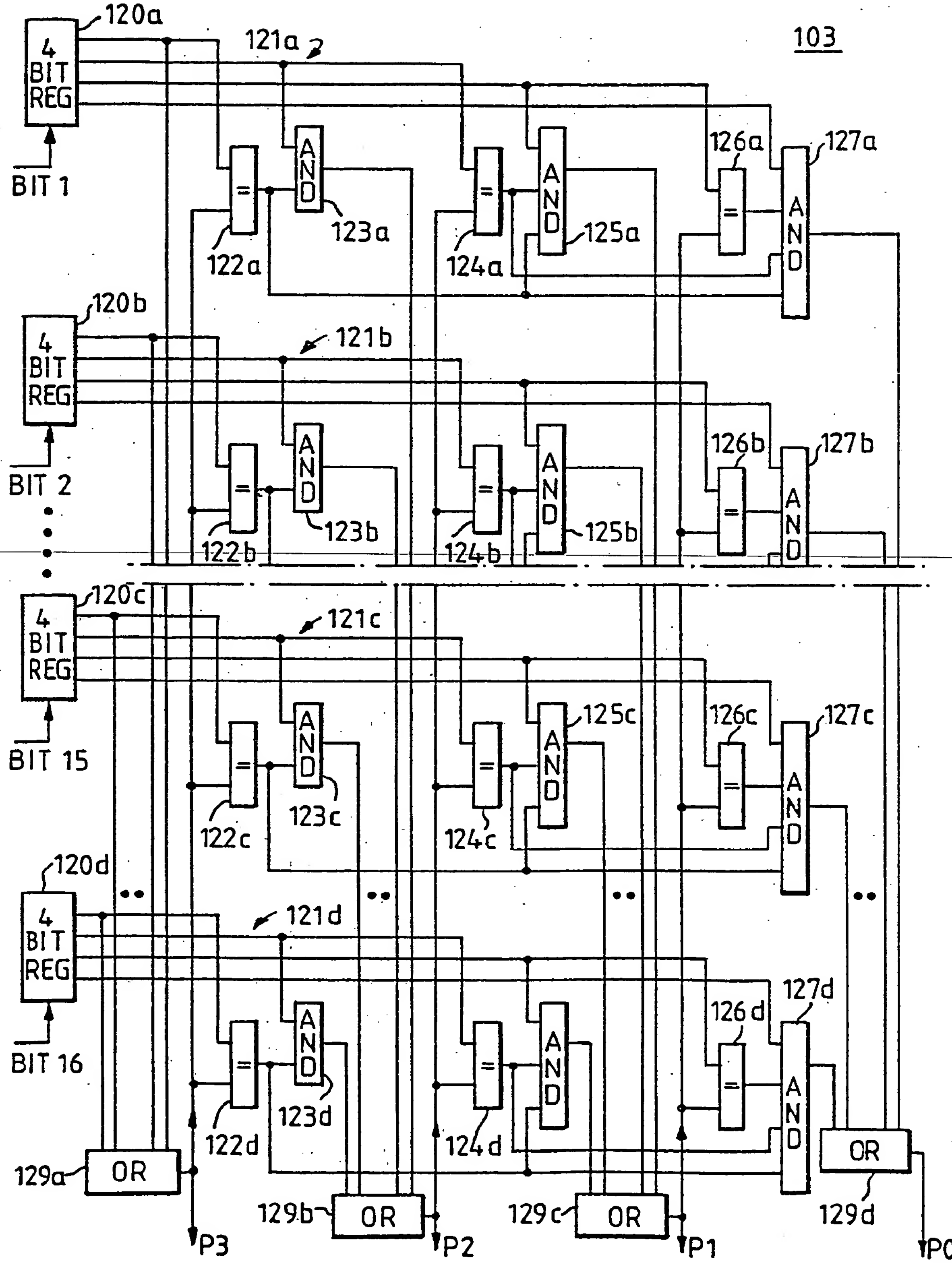

103
120a
121a
4 BIT REG
BIT 1
122a
123a
124a
125a
126a
127a
AND
=
120b
121b
BIT 2
122b
123b
124b
125b
126b
127b
120c
121c
BIT 15
122c
123c
124c
125c
126c
127c
120d
121d
BIT 16
122d
123d
124d
126d
127d
OR
129a
129b
129c
129d
P3
P2
P1
P0

FIG. 9

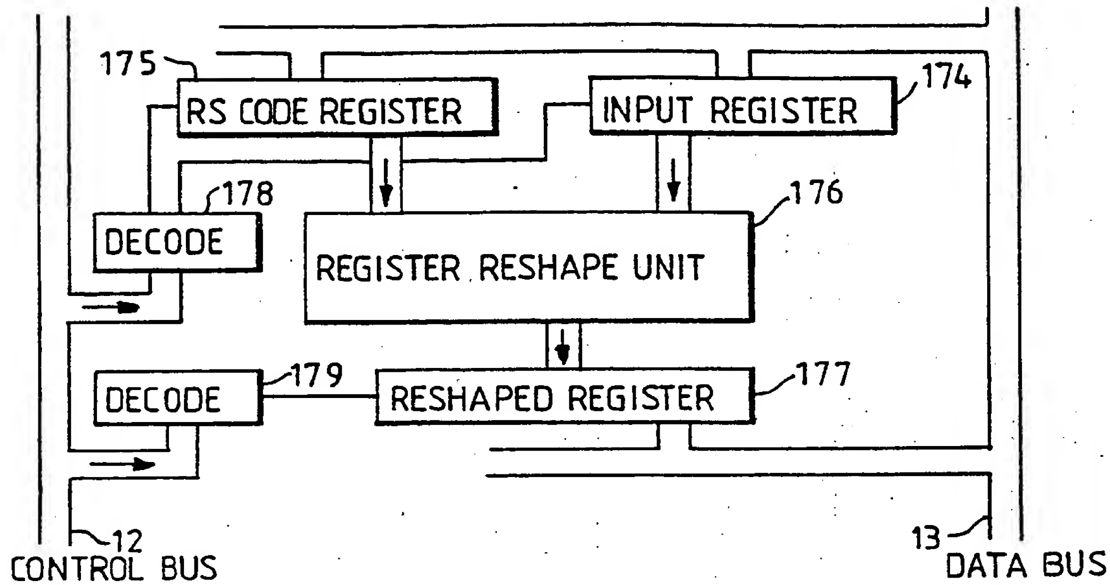

175
RS CODE REGISTER
INPUT REGISTER
174
178
DECODE
176
REGISTER RESHAPE UNIT
DECODE
179
RESHAPED REGISTER
177
12
CONTROL BUS
13
DATA BUS

FIG. 10

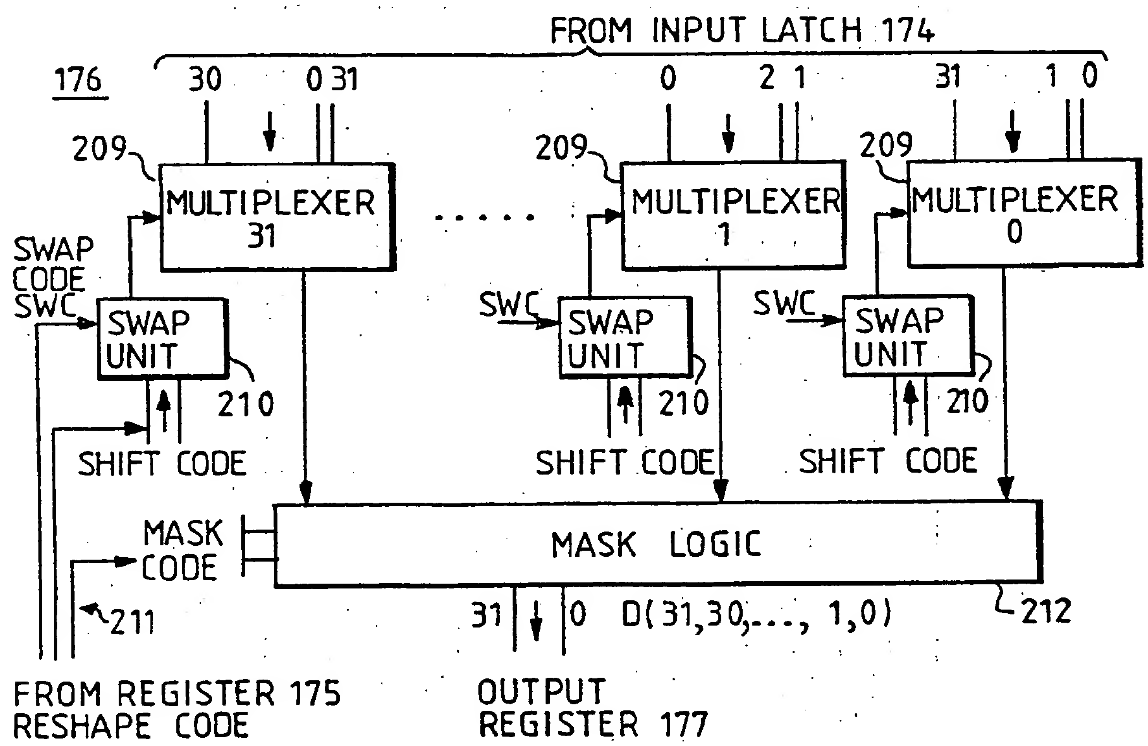

FROM INPUT LATCH 174
176
30 0 31
0 2 1
31 1 0
209
MULTIPLEXER 31
MULTIPLEXER 1
MULTIPLEXER 0
SWAP CODE SWC
SWC
SWAP UNIT
210
SHIFT CODE
MASK CODE
MASK LOGIC
211
31 0 D(31,30,..., 1,0)
212
FROM REGISTER 175
RESHAPE CODE
OUTPUT
REGISTER 177

FIG. 11

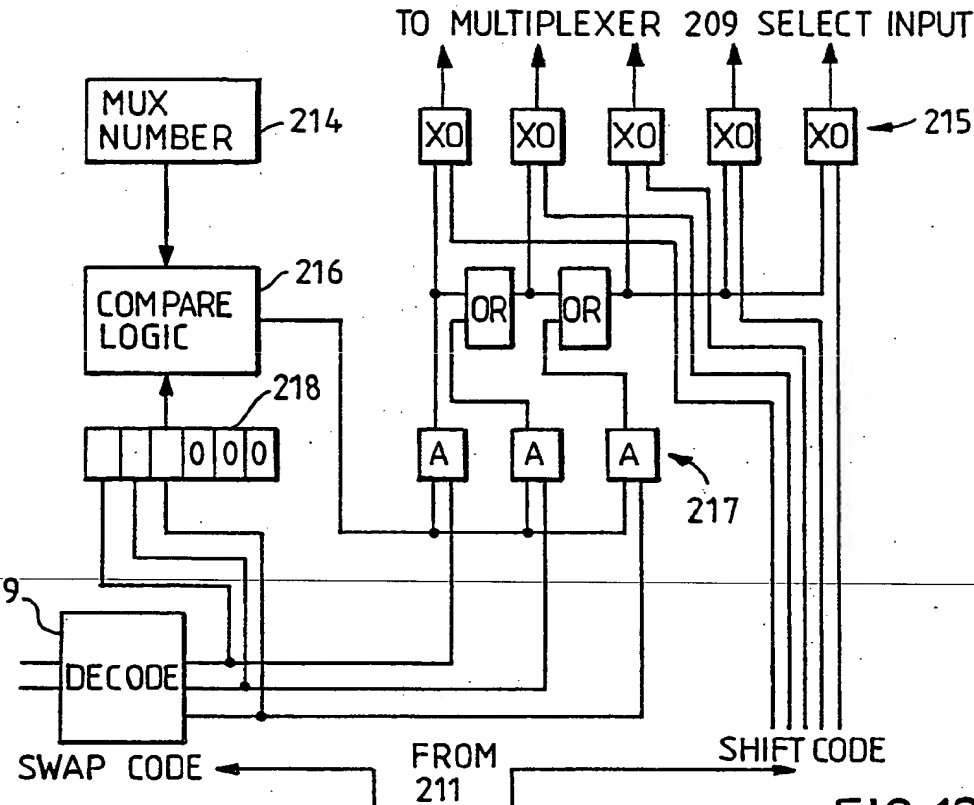

TO MULTIPLEXER 209 SELECT INPUT
MUX NUMBER
214
XO
XO
XO
XO
XO
215
COMPARE LOGIC
216
OR
OR
218
0
0
0
A
A
A
217
219
DECODE
SWAP CODE
FROM 211
SHIFT CODE

FIG. 12

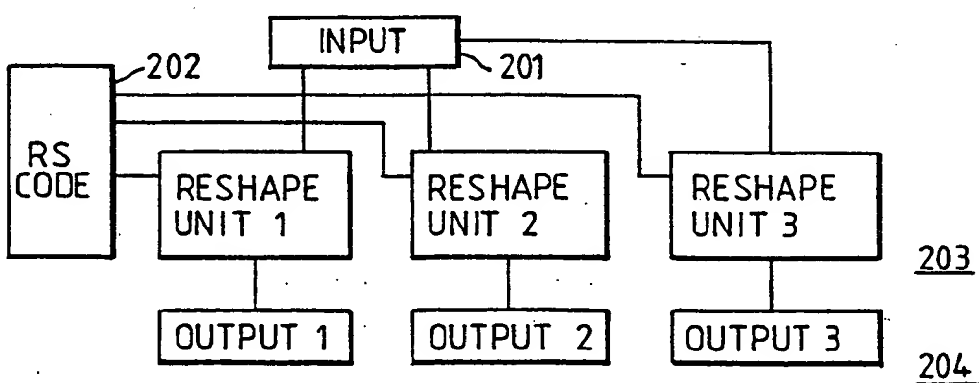

INPUT
202
201
RS CODE
RESHAPE UNIT 1
RESHAPE UNIT 2
RESHAPE UNIT 3
203
OUTPUT 1
OUTPUT 2
OUTPUT 3
204

FIG. 13

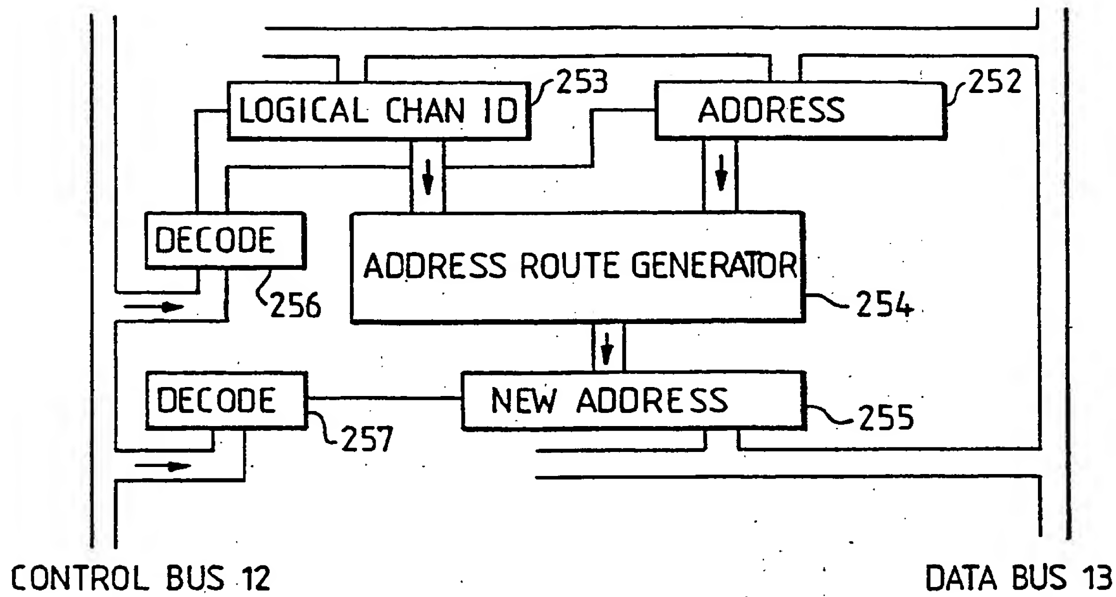

LOGICAL CHAN ID
253
ADDRESS
252
DECODE
256
ADDRESS ROUTE GENERATOR
254
DECODE
257
NEW ADDRESS
255
CONTROL BUS 12
DATA BUS 13

FIG. 14

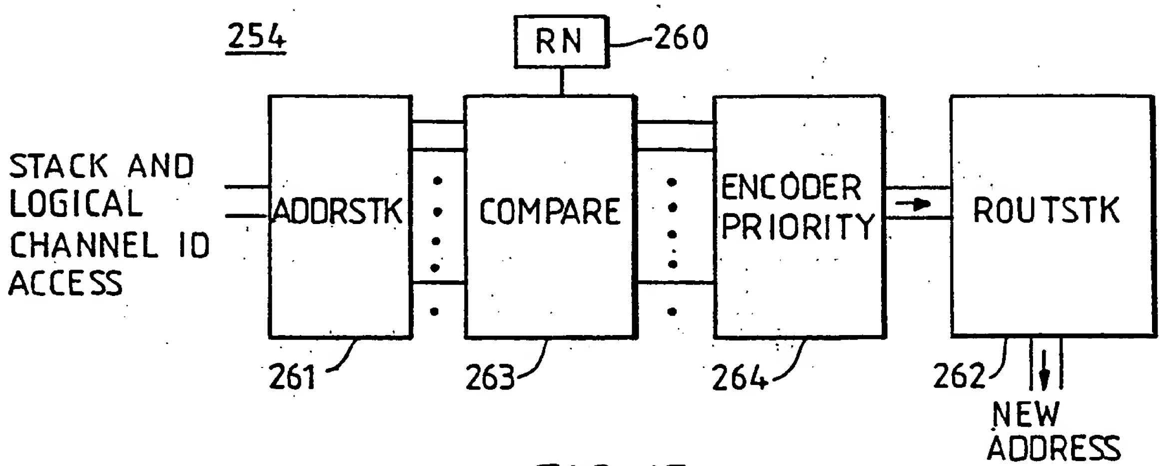

254
RN
260
STACK AND LOGICAL CHANNEL ID ACCESS
ADDRSTK
COMPARE
ENCODER PRIORITY
ROUTSTK
261
263
264
262
NEW ADDRESS

FIG. 15

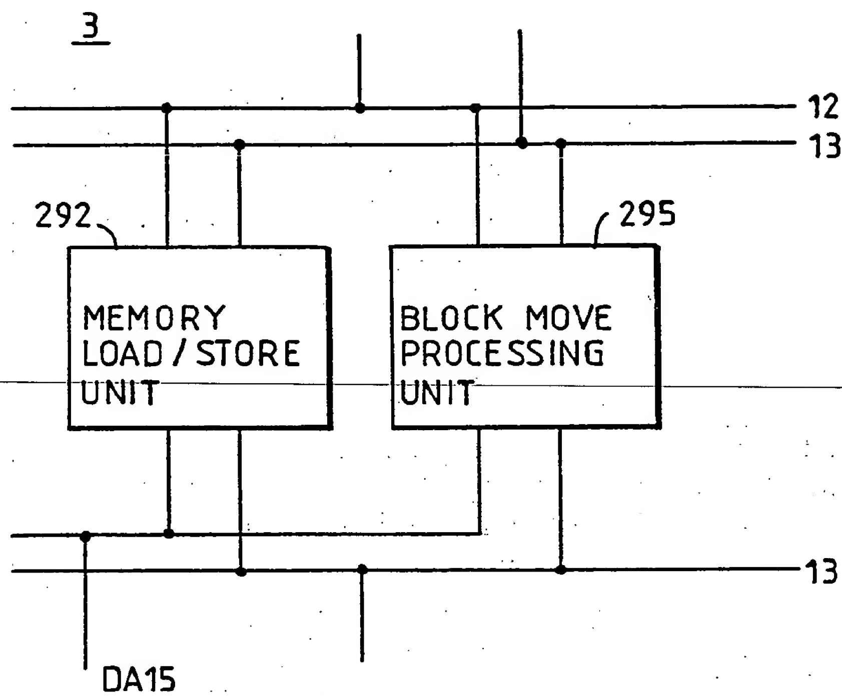

3
12
13
292
295
MEMORY
LOAD / STORE
UNIT
BLOCK MOVE
PROCESSING
UNIT
13
DA15

FIG. 16

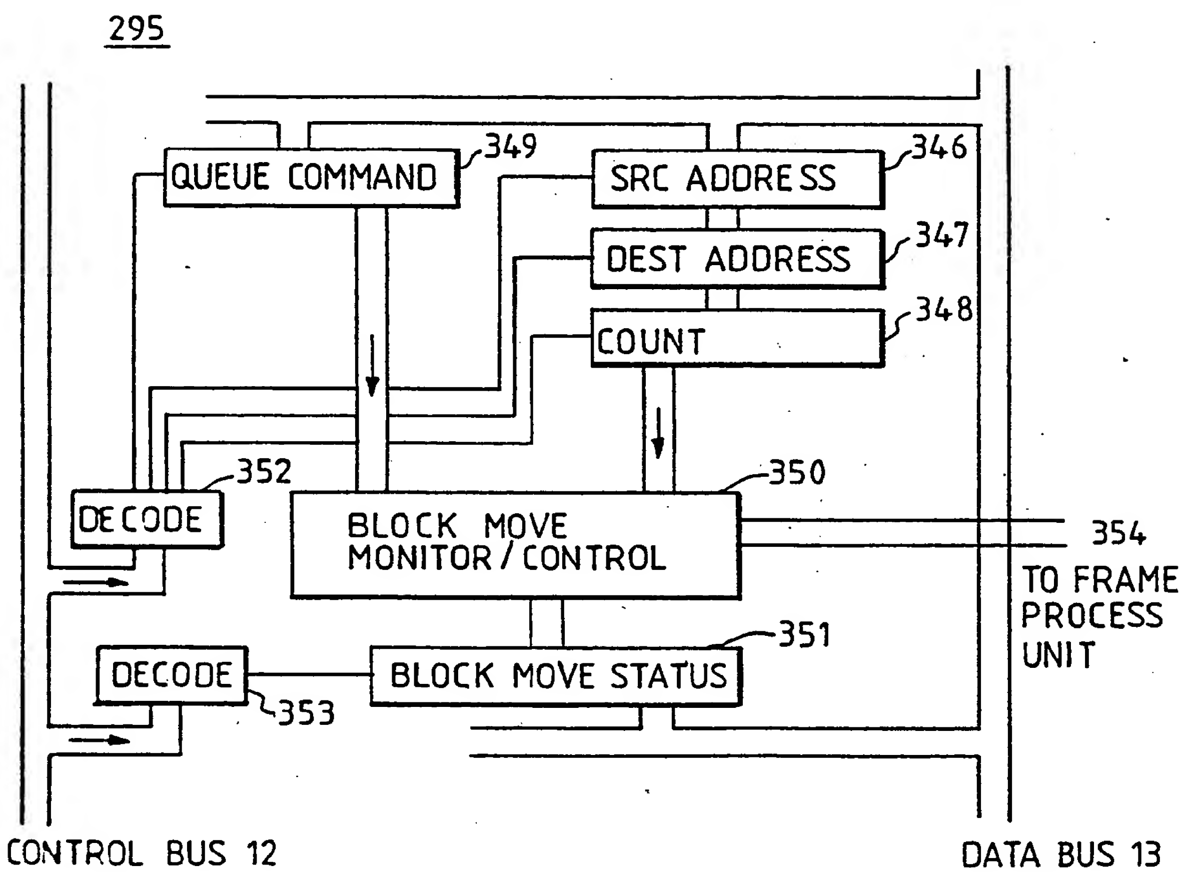

295
QUEUE COMMAND
349
SRC ADDRESS
346
DEST ADDRESS
347
COUNT
348
352
DECODE
350
BLOCK MOVE
MONITOR / CONTROL
354
TO FRAME
PROCESS
UNIT
351
DECODE
BLOCK MOVE STATUS
353
CONTROL BUS 12
DATA BUS 13

FIG. 17

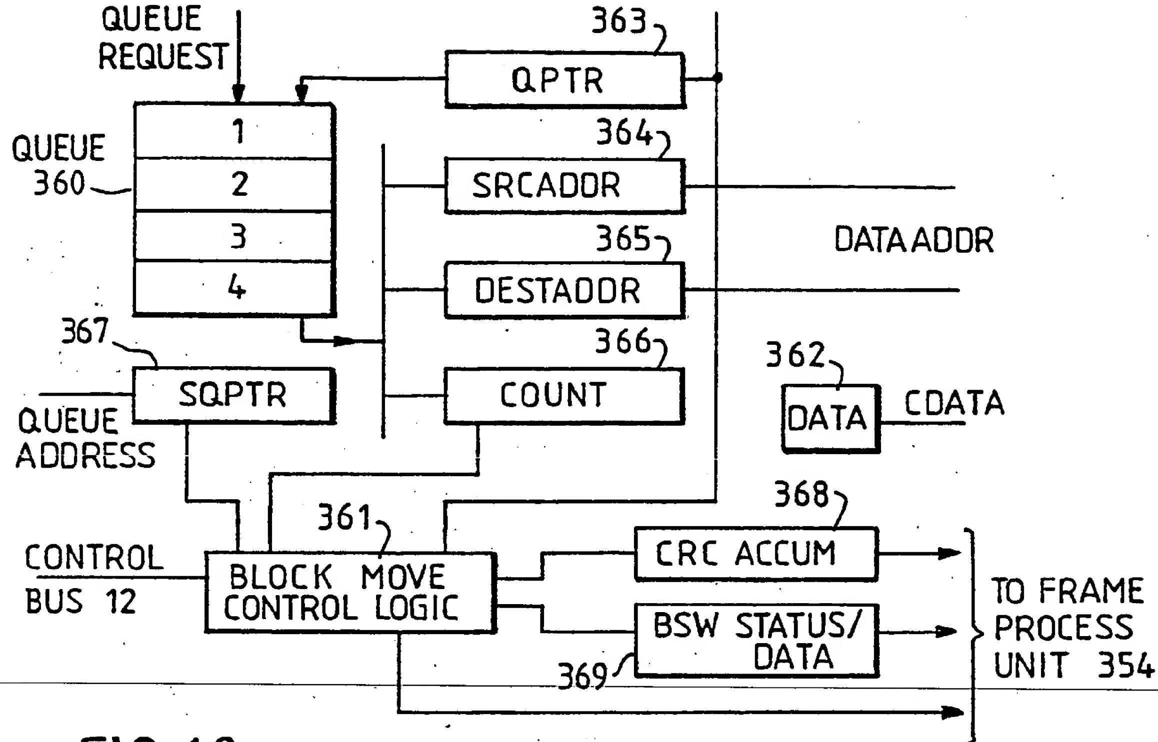

QUEUE
REQUEST
363
QPTR
QUEUE
360
1
2
3
4
364
SRCADDR
365
DESTADDR
DATA ADDR
366
COUNT
367
SQPTR
QUEUE
ADDRESS
362
DATA
CDATA
368
361
CONTROL
BUS 12
BLOCK MOVE
CONTROL LOGIC
CRC ACCUM
BSW STATUS/
DATA
369
TO FRAME
PROCESS
UNIT 354

FIG.18

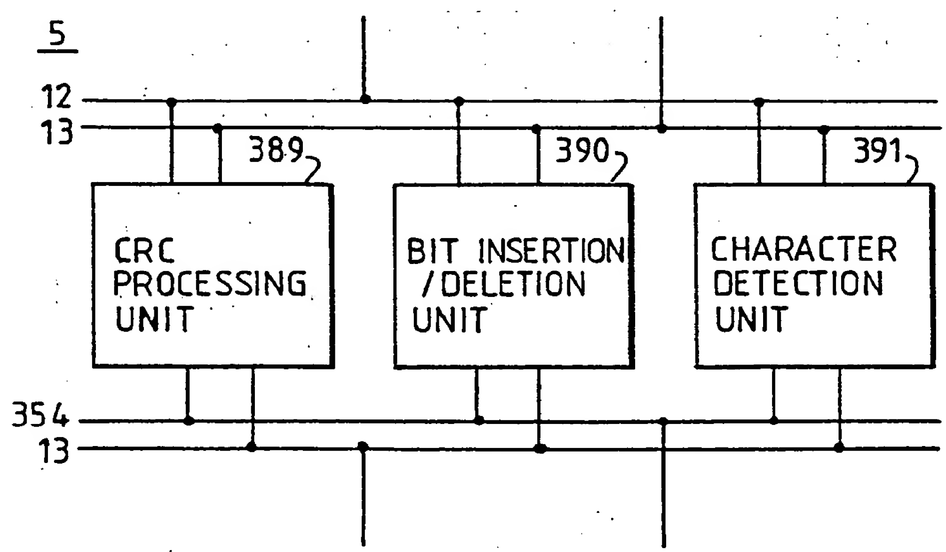

5
12
13
389
390
391
CRC
PROCESSING
UNIT
BIT INSERTION
/DELETION
UNIT
CHARACTER
DETECTION
UNIT
354
13

FIG.19

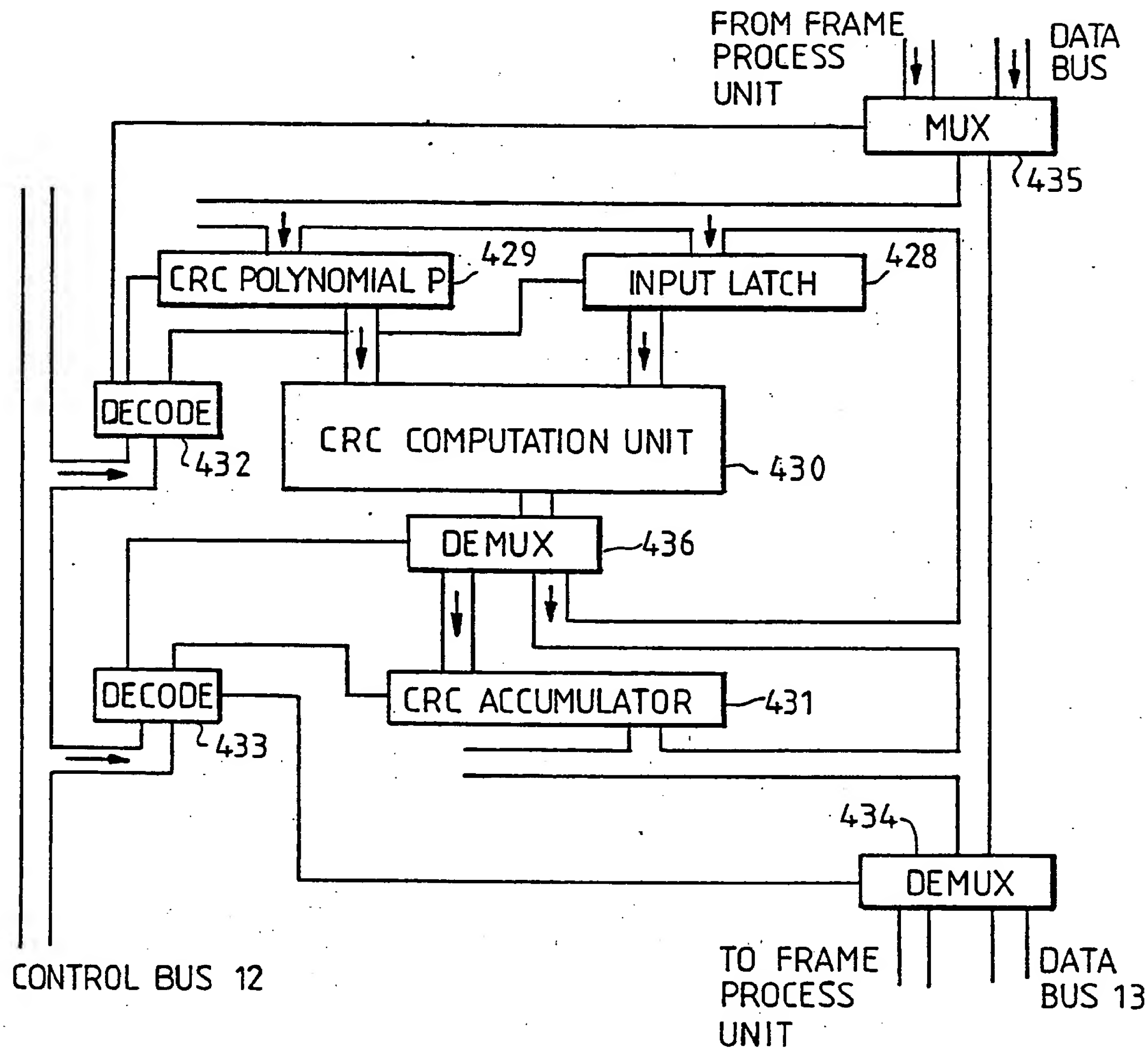

FROM FRAME
PROCESS
UNIT
DATA
BUS
MUX
435
429
428
CRC POLYNOMIAL P
INPUT LATCH
DECODE
432
CRC COMPUTATION UNIT
430
DEMUX
436
DECODE
433
CRC ACCUMULATOR
431
434
DEMUX
CONTROL BUS 12
TO FRAME
PROCESS
UNIT
DATA
BUS 13

FIG. 20

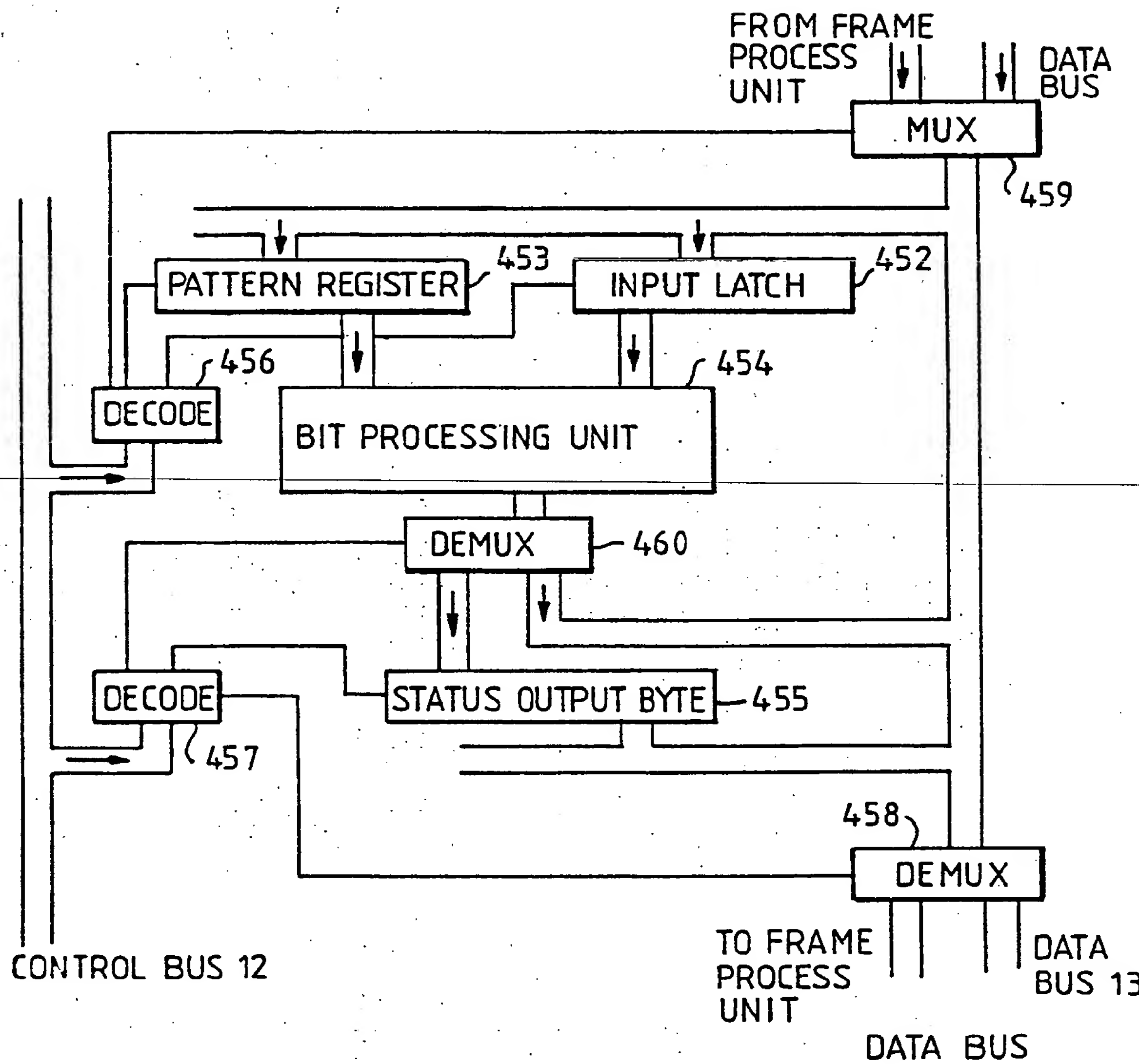

FROM FRAME PROCESS UNIT
DATA BUS
MUX
459
453
452
PATTERN REGISTER
INPUT LATCH
456
454
DECODE
BIT PROCESSING UNIT
DEMUX
460
DECODE
STATUS OUTPUT BYTE
455
457
458
DEMUX
CONTROL BUS 12
TO FRAME PROCESS UNIT
DATA BUS 13
DATA BUS

FIG. 21

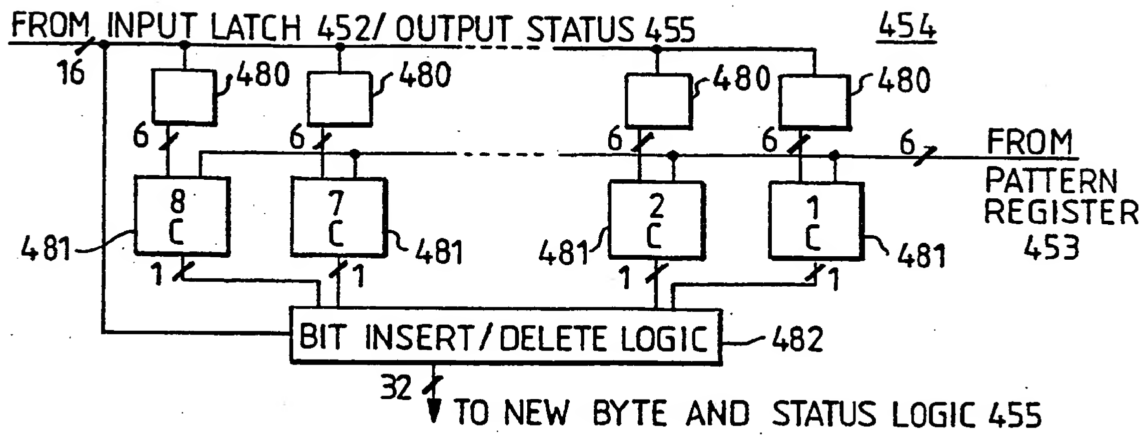


Table for Bit Insert/Delete Logic

		Input Bits		Output
T	C	PB	NB	OB
0	1	43210	76543210	P7654321
0	2	43210	76543210	7P654321
0	3	43210	76543210	76P54321
0	4	43210	76543210	765P4321
0	5	43210	76543210	7654P321
0	6	43210	76543210	76543P21
0	7	43210	76543210	765432P1
0	8	43210	76543210	7654321P
1	1	43210	76543210	
1	2	43210	76543210	07543210
1	3	43210	76543210	10764321
1	4	43210	76543210	21076532
1	5	43210	76543210	32107654
1	6	43210	76543210	43210765
1	7	43210	76543210	54321076
1	8	43210	76543210	65432107

PART 1

Present Status		Next Status		
Count	Residue	Count	Residue	E
000		001	0	0
001	X	010	0X	0
010	XX	011	0XX	0
011	XXX	100	0XXX	0
100	XXXX	101	0XXXX	0
101	XXXXX	110	0XXXXX	0
110	XXXXXX	111	0XXXXXX	0
111	XXXXXXX	000	0XXXXXXX	1
000		111	6453210	1
001	0	000		0
010	10	001	0	0
011	210	010	10	0
100	3210	011	210	0
101	43210	100	4310	0
110	543210	101	54320	0
111	6543210	110	654321	0

PART 2

where
T = type of operation Insertion=0/Deletion=1
C = Comparator (C) output number active
PB = 5 LSB's of previous byte, NB = New byte, OB = Output Byte
P = LSB of 6 bit pattern in pattern register for insertion
Count = Number of outstanding bits less than 8 after insertion or deletion
Residue = The actual bits '0XX..X' before and after insertion where '0' is inserted bit from new byte
E = 1 if extra eight bits completed due to insertion or deletion count is eight bits

Assumptions
1) Part 1 may contain two comparator (C) output values to be active in which case the count and residue would change by two bits.
2) Part 2 is shown separately because the count is independent of the active comparator (C) number and that the correspondence is only valid for the output bits in Part 1 to the residue bits in part 2.

FIG. 22

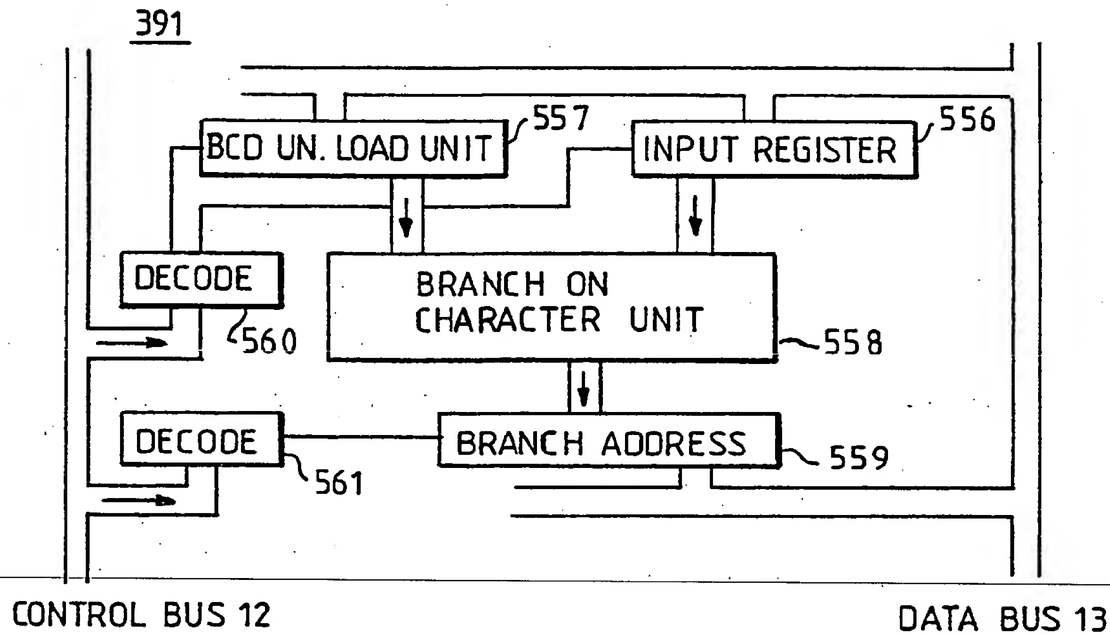

391
BCD UN. LOAD UNIT
557
INPUT REGISTER
556
DECODE
560
BRANCH ON CHARACTER UNIT
558
DECODE
561
BRANCH ADDRESS
559
CONTROL BUS 12
DATA BUS 13

FIG. 23

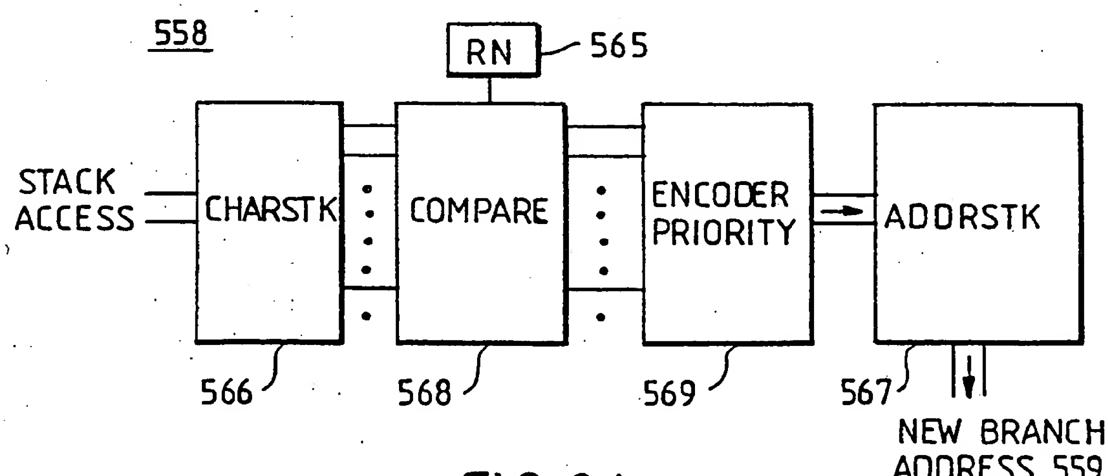

558
RN
565
STACK ACCESS
CHARSTK
COMPARE
ENCODER PRIORITY
ADDRSTK
566
568
569
567
NEW BRANCH ADDRESS 559

FIG. 24

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 363 173 A3**

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number: 89310147.7

(51) Int. Cl.5: **G06F 13/12**, H04L 29/08

(22) Date of filing: **04.10.89**

(30) Priority: **07.10.88 US 254986**

(43) Date of publication of application: **11.04.90 Bulletin 90/15**

(84) Designated Contracting States: **DE FR GB IT**

(88) Date of deferred publication of the search report: **24.07.91 Bulletin 91/30**

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Davis, Gordon Taylor**
1285 West Royal Palm Road
Boca Raton Florida 33486(US)
Inventor: **Mandalia, Baiju Dhirajlal**
9900 Baywater Drive
Boca Raton Florida 33496(US)
Inventor: **Landa, Robert Egugene**
7159 Northwest 3rd Avenue
Boca Raton Florida 33487(US)
Inventor: **Van den Berg, Jan Wouter**
541 Phillips Drive
Boca Raton Florida 33432(US)
Inventor: **Van Voorhis, David Curtis**
1225 Southwest 21st Street
Boca Raton Florida 33486(US)

(74) Representative: **Grant, Iain Murray**
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN(GB)

(54) **Communication processors for layered protocols.**

(57) A special purpose communications protocol processor (CPP) provides more efficient processing of layered communication protocols -- e.g. SNA (Systems Network Architecture) and OSI (Open Systems Interconnection) -- than contemporary general purpose processors, permitting hitherto unavailable operations relative to high speed communication links. The CPP contains special-purpose circuits dedicated to quick performance (e.g. single machine cycle execution) of functions needed to process header and frame information, such functions and information being of the sort repeatedly encountered in all protocol layers, and uses instructions architected to operate these circuits. The header processing functions given special treatment in this manner include priority branch determination functions, register bit reshaping (rearranging) functions, and instruction address processing functions. Frame processing functions so handled include CRC (cyclic redundancy check) computations, bit insertion/deletion operations and special character detection operations.

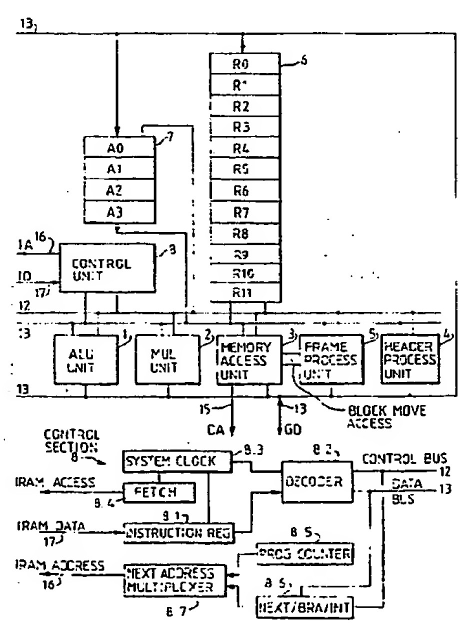


FIG. 1

EP 0 363 173 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number

EP 89 31 0147

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
A	MICROPROCESSING AND MICROPROGRAMMING. vol. 24, 1988, AMSTERDAM NL pages 103 - 110; M.DANG et al: "SPECIFIC DATA STRUCTURE INTENDED FOR THE IMPLEMENTATION OF HIGH LEVEL ISO STANDARDS:ASSOCIATED ALGORITHMS AND DEDICATED HARDWARE" * the whole document *	1,14	G 06 F 13/12 H 04 L 29/08
A	PROCEEDINGS IEEE INFOCOM'88,March 27-31,1988; NEW YORK US;pages 1072-1081;M.STREETHARAN et al: "ARCHITECTURE OF A MULTIPROCESSOR-BASED HIGH SPEED COMMUNICATIONS PROCESSOR" * page 1075,left hand column,line 41-page 1077, left hand column,line 19 *	1	
A	SPRING COMPCON 86,March 3-6,1986;NEW YORK US; pages 126-131;J.MOUSSOURIS et al:"A CMOS RISC PROCESSOR WITH INTEGRATED SYSTEM FUNCTIONS" * page 126,right hand column,line 20-page 128, right hand column,line 4;figs 1-5 *	1	

TECHNICAL FIELDS SEARCHED (Int. Cl.5)

G 06 F

The present search report has been drawn up for all claims

Place of search	Date of completion of search	Examiner
The Hague	15 May 91	WANZEELE R.J.

CATEGORY OF CITED DOCUMENTS

X: particularly relevant if taken alone
Y: particularly relevant if combined with another document of the same catagory
A: technological background
O: non-written disclosure
P: intermediate document
T: theory or principle underlying the invention
E: earlier patent document, but published on, or after the filing date
D: document cited in the application
L: document cited for other reasons
&: member of the same patent family, corresponding document